EtherCAT Specification – Part 6

Application Layer protocol specification

Document: ETG.1000.6 S (R) V1.0.4

Nomenclature:

ETG-Number	ETG.1000.6
Type	S (Standard)
State	R (Release)
Version	V1.0.4

Created by: ETG
Contact: info@ethercat.org
Date 15.09.2017

Trademarks and Patents

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

Disclaimer

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

Copyright

DOCUMENT HISTORY

Version	Comment
1.0	Adopted from IEC-Standard 61158-6 Type 12 for ETG use only!
1.0.1	Corrections and clarifications according to ETG1000_ES_D_V0i6_EcatSpecErrata.pdf
1.0.2	Corrections and clarifications according to ETG1000_V1i0i2_ES_R_V0i7_EcatSpecErrata.doc and IEC SC65C MT 9 editorial comments
1.0.3	Corrections and clarifications according to ETG1000_V1i0i2_ES_R_V0i8_EcatSpecErrata.doc
1.0.4	Corrections and clarifications according to ETG1000_V1i0i3_ES_R_V0i9_EcatSpecErrata.doc

CONTENTS

Figures

Tables

1 Scope

1.1 Scope of this standard and accordance to IEC Standards

The ETG.1000 series specifies the EtherCAT Technology within the EtherCAT Technology Group. It is divided into the following parts:

- ETG.1000.2: Physical Layer service definition and protocol specification
- ETG.1000.3: Data Link Layer service definition
- ETG.1000.4: Data Link Layer protocol specification
- ETG.1000.5: Application Layer service definition
- ETG.1000.6: Application Layer protocol specification

These parts are based on the corresponding parts of the IEC 61158 series Type 12. EtherCAT is named Type 12 in IEC 61158 to avoid the usage of brand names.

1.2 General

The fieldbus Application Layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a “window between corresponding application programs.”

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to EtherCAT fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard defines in an abstract way the externally visible behavior provided by the different Types of the fieldbus Application Layer in terms of

a) the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,

b) the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,

c) the application context state machine defining the application service behavior visible between communicating application entities; and

d) the application relationship state machines defining the communication behavior visible between communicating application entities; and.

The purpose of this standard is to define the protocol provided to

1) define the wire-representation of the service primitives defined in ETG.1000.5, and
2) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI Application Layer Structure (ISO/IEC 9545).

FAL services and protocols are provided by FAL application-entities (AE) contained within the application processes. The FAL AE is composed of a set of object-oriented Application Service Elements (ASEs) and a Layer Management Entity (LME) that manages the AE. The ASEs provide communication services that operate on a set of related application process object (APO) classes. One of the FAL ASEs is a management ASE that provides a common set of services for the management of the instances of FAL classes.

Although these services specify, from the perspective of applications, how request and responses are issued and delivered, they do not include a specification of what the requesting and responding applications are to do with them. That is, the behavioral aspects of the applications are not specified; only a definition of what requests and responses they can send/receive is specified. This permits greater flexibility to the FAL users in standardizing such object behavior. In addition to these services, some supporting services are also defined in this standard to provide access to the FAL to control certain aspects of its operation.

1.3 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in ETG.1000.5.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols.

1.4 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

There is no conformance of equipment to the application layer service definition standard. Instead, conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60559, *Binary Floating-point Arithmetic for Microprocessor Systems*

ISO/IEC 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model – Part1: The Basic Model*

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model – Part3: Naming and addressing*

ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems - Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 9899, Programming Languages – C*.*

ISO/IEC 10731, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

IEEE 802.1D, 2004, *IEEE standard for local and metropolitan area networks – Common specifications – Media access control (MAC) Bridges;* available at <http://www.ieee.org>

IEEE 802.1Q, 1998, *IEEE standard for Local and metropolitan area networks – Virtual bridged local area networks Bridges;* available at <http://www.ieee.org>

IETF RFC 768, *User Datagram Protocol*; available at <http://www.ietf.org>

IETF RFC 791, *Internet Protocol darpa internet program protocol specification*; available at <http://www.ietf.org>

3 Terms, definitions, symbols, abbreviations and conventions

For the purposes of this document, the following terms, definitions, symbols, abbreviations and conventions apply.

3.1 Reference model terms and definitions

This standard is based in part on the concepts developed in ISO/IEC 7498-1 and ISO/IEC 7498-3, and makes use of the following terms defined therein:

3.1.1 correspondent (N)-entities [7498-1]
correspondent AL-entities (N=7)

3.1.2 (N)-entity [7498-1]
AL-entity (N=7)

3.1.3 (N)-layer [7498-1]
AL-layer (N=7)

3.1.4 layer-management [7498-1]

3.1.5 peer-entities [7498-1]

3.1.6 primitive name [7498-3]

3.1.7 AL-protocol [7498-1]

3.1.8 AL-protocol-data-unit [7498-1]

3.1.9 reset [7498-1]

3.1.10 routing [7498-1]

3.1.11 segmenting [7498-1]

3.1.12 (N)-service [7498-1]
AL-service (N=7)

3.1.13 AL-service-data-unit [7498-1]

3.1.14 AL-simplex-transmission [7498-1]

3.1.15 AL-subsystem [7498-1]

3.1.16 systems-management [7498-1]

3.1.17 AL-user-data [7498-1]

3.2 Service convention terms and definitions

This standard also makes use of the following terms defined in ISO/IEC 10731 as they apply to the data-link layer:

3.2.1 acceptor

3.2.2 asymmetrical service

3.2.3 confirm (primitive);
requestor.deliver (primitive)

3.2.4 deliver (primitive)

3.2.5 AL-service-primitive;
primitive

3.2.6 AL-service-provider

3.2.7 AL-service-user

3.2.8 indication (primitive); acceptor.deliver (primitive)

3.2.9 request (primitive); requestor.submit (primitive)

3.2.10 requestor

3.2.11 response (primitive); acceptor.submit (primitive)

3.2.12 submit (primitive)

3.2.13 symmetrical service

3.3 Application layer definitions

3.3.1
application
function or data structure for which data is consumed or produced

3.3.2
application objects
multiple object classes that manage and provide a run time exchange of messages across the network and within the network device

3.3.3
application process
part of a distributed application on a network, which is located on one device and unambiguously addressed

3.3.4
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation. This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities

3.3.5
attribute
description of an externally visible characteristic or feature of an object

NOTE The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.

3.3.6
basic slave
slave device that supports only physical addressing of data

3.3.7
behavior
indication of how an object responds to particular events

3.3.8
bit
unit of information consisting of a 1 or a 0. This is the smallest data unit that can be transmitted

3.3.9
channel
representation of a single physical or logical management object of a slave to control conveyance of data

3.3.10
client
1) object which uses the services of another (server) object to perform a task

2) initiator of a message to which a server reacts

3.3.11
clock synchroization
representation of a sequence of interactions to synchronize the clocks of all time receivers by a time master

3.3.12
communication object
component that manages and provides a run time exchange of messages across the network

3.3.13
connection
logical binding between two application objects within the same or different devices

3.3.14
consume
act of receiving data from a provider

3.3.15
consumer
node or sink receiving data from a provider

3.3.16
conveyance path
unidirectional flow of APDUs across an application relationship

3.3.17
cyclic
events which repeat in a regular and repetitive manner

3.3.18
data
generic term used to refer to any information carried over a fieldbus

3.3.19
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.3.20
data type
relation between values and encoding for data of that type according to the definitions of IEC 61131-3.

3.3.21
data type object
entry in the object dictionary indicating a data type

3.3.22
default gateway
device with at least two interfaces in two different IP subnets acting as router for a subnet.

3.3.23
device
physical entity connected to the fieldbus composed of at least one communication element (the network element) and which may have a control element and/or a final element (transducer, actuator, etc.)

3.3.24
device profile
collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.3.25
diagnosis information
all data available at the server for maintenance purposes

3.3.26
distributed clocks
method to synchronize slaves and maintain a global time base

3.3.27
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.3.28
error class
general grouping for related error definitions and corresponding error codes

3.3.29
error code
identification of a specific type of error within an error class

3.3.30
event
instance of a change of conditions

3.3.31
fieldbus memory management unit
function that establishes one or several correspondences between logical addresses and physical memory

3.3.32
fieldbus memory management unit entity
single element of the fieldbus memory management unit: one correspondence between a coherent logical address space and a coherent physical memory location

3.3.33
frame
denigrated synonym for DLPDU

3.3.34
full slave
slave device that supports both physical and logical addressing of data

3.3.35
index
address of an object within an application process

3.3.36
interface
shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

3.3.37
little endian
data representation of multi-octet fields where the least significant octet is transmitted first.

3.3.38
master
device that controls the data transfer on the network and initiates the media access of the slaves by sending messages and that constitutes the interface to the control system

3.3.39
mapping
correspondence between two objects in that way that one object is part of the other object

3.3.40
mapping parameters
set of values defining the correspondence between application objects and process data objects

3.3.41
medium
cable, optical fibre or other means by which communication signals are transmitted between two or more points

NOTE "media" is the plural of medium.

3.3.42
message
ordered series of octets intended to convey information

NOTE Normally used to convey information between peers at the application layer.

3.3.43
network
set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.3.44
node

a) single DL-entity as it appears on one local link

b) end-point of a link in a network or a point at which two or more links meet [derived from IEC 61158-2]

3.3.45
object
abstract representation of a particular component within a device

NOTE An object can be

1) an abstract representation of the capabilities of a device. Objects can be composed of any or all of the following components:

a) data (information which changes with time);

b) configuration (parameters for behavior);

c) methods (things that can be done using data and configuration).

2) a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior.

3.3.46
object dictionary
data structure addressed by Index and Sub-index that contains description of data type objects, communication objects and application objects

3.3.47
process data
collection of application objects designated to be transferred cyclically or acyclically for the purpose of measurement and control

3.3.48
process data object
structure described by mapping parameters containing one or several process data entities

3.3.49
producer
node or source sending data to one or many consumers

3.3.50
resource
processing or information capability

3.3.51
segment
collection of one real master with one or more slaves

3.3.52
server
object which provides services to another (client) object

3.3.53
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.3.54
slave
DL-entity accessing the medium only after being initiated by the preceding slave or the master

3.3.55
subindex
sub-address of an object within the object dictionary

3.3.56
sync manager
collection of control elements to coordinate access to concurrently used objects

3.3.57
sync manager channel
single control elements to coordinate access to concurrently used objects

3.3.58
switch
MAC bridge as defined in IEEE 802.1D

3.4 Common symbols and abbreviations

3.4.1 AL-	Application layer (as a prefix)
3.4.2 ALE	AL-entity (the local active instance of the application layer)
3.4.3 AL	AL-layer
3.4.4 APDU	AL-protocol-data-unit
3.4.5 ALM	AL-management
3.4.6 ALME	AL-management Entity (the local active instance of AL-management)
3.4.7 ALMS	AL-management service
3.4.8 ALS	AL-service
3.4.9 ALSDU	AL-service-data-unit
3.4.10 DL	Data-link-layer
3.4.11 FIFO	First-in first-out (queuing method)
3.4.12 OSI	Open systems interconnection
3.4.13 PhL	Ph-layer
3.4.14 QoS	Quality of service

3.5 Additional symbols and abbreviations

3.5.1 AoE	Automation Device Specification (ADS) over EtherCAT
3.5.2 ASE	Application service element
3.5.3 AR	Application relationship
3.5.4 CAN	Controller Area Network
3.5.5 CiA	CAN in Automation
3.5.6 CoE	CAN application protocol over EtherCAT services
3.5.7 CSMA/CD	Carrier sense multiple access with collision detection
3.5.8 DC	Distributed clocks
3.5.9 DNS	Domain name system (server for name resolution in IP networks)
3.5.10 E^{2}PROM	Electrically erasable programmable read only memory
3.5.11 EoE	Ethernet tunneled over EtherCAT services
3.5.12 ESC	EtherCAT slave controller
3.5.13 FCS	Frame check sequence
3.5.14 FMMU	Fieldbus memory management unit
3.5.15 FoE	File access with EtherCAT services
3.5.16 HDR	Header
3.5.17 ID	Identifier
3.5.18 IETF	Internet engineering rask force
3.5.19 IP	Internet protocol
3.5.20 LAN	Local area network
3.5.21 MAC	Medium access control
3.5.22 OD	Object dictionary
3.5.23 PDI	Physical device internal interface (a set of elements that allows access to DL services from the AL)
3.5.24 PDO	Process data object
3.5.25 RAM	Random access memory
3.5.26 Rx	Receive

3.5.27 SDO Service data object

3.5.28 SII slave information interface

3.5.29 SM Synchronization manager

3.5.30 SoE Servo drive profile with EtherCAT services

3.5.31 SyncM Synchronization manager

3.5.32 TCP Transmission control protocol

3.5.33 Tx Transmit

3.5.34 UDP User datagram protocol

3.5.35 VoE Profile specific services

3.5.36 WKC Working counter

3.6 Conventions

3.6.1 General concept

The services are specified in ETG.1000.3 standard. The service specification defines the services that are provided by the EtherCAT DL. The mapping of these services to ISO/IEC 8802-3 is described in this Standard.

This standard uses the descriptive conventions given in ISO/IEC 10731.

3.6.2 Convention for the encoding of reserved bits and octets

The term "reserved" may be used to describe bits in octets or whole octets. All bits or octets that are reserved should be set to zero at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved bits or octets are checked by a state machine.

The term "reserved" may also be used to indicate that certain values within the range of a parameter are reserved for future extensions. In this case the reserved values should not be used at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved values are check by a state machine.

3.6.3 Conventions for the common codings of specific field octets

APDUs may contain specific fields that carry information in a primitive and condensed way. These fields shall be coded in the order according to Figure 1.

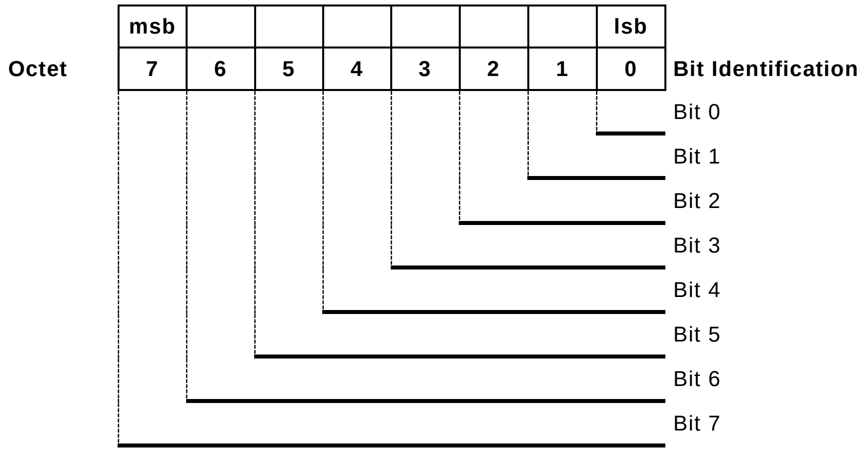

Figure 1 – Common structure of specific fields

Bits may be grouped as group of bits. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. Alias names may be used for each bit or group of bits or they may be marked as reserved. The grouping of individual bits shall be in ascending order without gaps. The values for a group of bits may be represented as binary, decimal or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 8 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest number of the group represents the msb concerning the grouped bits.

EXAMPLE 1: Description and relation for the specific field octet

Bit 0: reserved.

Bit 1-3: Reason_Code The decimal value 2 for the Reason_Code means general error.

Bit 4-7: shall always set to one.

The octet that is constructed according to the description above looks as follows:

(msb) Bit 7 = 1,

Bit 6 = 1,

Bit 5 = 1,

Bit 4 = 1,

Bit 3 = 0,

Bit 2 = 1,

Bit 1 = 0,

(lsb) Bit 0 = 0.

This bit combination has an octet value representation of 0xf4.

3.6.4 Abstract syntax conventions

The AL syntax elements related to PDU structure are described as shown in the example of Table 1.

Frame part denotes the element that will be replaced by this reproduction.

Data field is the name of the elements.

Data Type denotes the type of the terminal symbol.

Value/Description contains the constant value or the meaning of the parameter.

Table 1 – PDU element description example

Frame part	Data Field	Data Type	Value/Description
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Size Indicator	Unsigned1	0x00: size of Data (1..4) unspecified 0x01: size of Data in Data Set Size specified
	Transfer Type	Unsigned1	0x01: Expedited transfer
	Data Set Size	Unsigned2	0x00: 4 Octet Data 0x01: 3 Octet Data 0x02: 2 Octet Data 0x03: 1 Octet Data
	Complete Access	Unsigned1	0x00
	Command Specifier	Unsigned3	0x01: Initiate Download Request

The informational attribute types are described in C language notations (ISO/IEC 9899) as shown in Figure 2. BYTE and WORD are elements of type unsigned char and unsigned short.

```
typedef struct
{
  Unsigned8     Type;
  Unsigned8     Revision;
  Unsigned16    Build;
  Unsigned8     NoOfSuppFmmuChannels;
  Unsigned8     NoOfSuppSyncManChannels;
  Unsigned8     RamSize;
  Unsigned8     Reserved1;
  unsigned      FmmuBitOperationNotSupp:  1;
  unsigned      Reserved2:                7;
  unsigned      Reserved3:                8;
} TDLINFORMATION;
```

Figure 2 **– Type description example**

The attributes of an object are described in a form as shown in Table 2.

Subindex describes a single element of the object.

Description denotes a name string for this attribute.

Data Type denotes the type of this element.

M/O/C indicates whether the attribute is mandatory (M), optional (O) or depends upon setting of other attributes (C).

Access type shows the access right to this element. R means read access right, W means write access right. It can be extended showing the AL state where the access right applies.

PDO Mapping denotes the possibility to map this attribute to TxPDO or RxPDO or to indicate that this parameter is not mappable.

Value contains the constant value and/or the meaning of the parameter.

Table 2 – Example attribute description

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of entries	UNSIGNED8	M	R	No	4
1	Vendor ID	UNSIGNED32	M	R	No	Assigned uniquely by ETG
2	Product Code	UNSIGNED32	M	R	No	Assigned uniquely by Vendor
3	Revision Number	UNSIGNED32	M	R	No	Assigned uniquely by Vendor
4	Serial Number	UNSIGNED32	M	R	No	assigned uniquely for this device by Vendor

3.6.5 State machine conventions

The protocol sequences are described by means of State Machines.

In state diagrams states are represented as boxes and state transitions are shown as arrows. Names of states and transitions of the state diagram correspond to the names in the textual listing of the state transitions.

The textual listing of the state transitions is structured as follows, see also Table 3.

The first column contains the name of the transition.

The second column defines the current state.

The third column contains an optional event followed by Conditions starting with a “/” as first line character and finally followed by the Actions starting with a “=>” as first line character.

The last column contains the next state.

If the event occurs and the conditions are fulfilled the transition fires, i.e. the actions are executed and the next state is entered.

The layout of a Machine description is shown in Table 3. The meaning of the elements of a State Machine Description is shown in Table 4.

Table 3 – State machine description elements

#	Current state	Event /Condition => Action	Next state

Table 4 – Description of state machine elements

Description element	Meaning
Current state Next state	Name of the given states.
#	Name or number of the state transition.
Event	Name or description of the event.
/Condition	Boolean expression. The preceding “\” is not part of the condition.
=> Action	List of assignments and service or function invocations. The preceding “=>” is not part of the action.

The conventions used in the state machines are shown in Table 5.

Table 5 – Conventions used in state machines

Convention	Meaning
=	Value of an item on the left is replaced by value of an item on the right. If an item on the right is a parameter, it comes from the primitive shown as an input event.
axx	A parameter name if a is a letter. Example: Identifier = reason means value of a 'reason' parameter is assigned to a parameter called 'Identifier.'
"xxx"	Indicates fixed visible string. Example: Identifier = "abc" means value "abc" is assigned to a parameter named 'Identifier.'
nnn	if all elements are digits, the item represents a numerical constant shown in decimal representation
0xnn	if all elements nn are digits, the item represents a numerical constant shown in hexadecimal representation
==	A logical condition to indicate an item on the left is equal to an item on the right.
<	A logical condition to indicate an item on the left is less than the item on the right.
>	A logical condition to indicate an item on the left is greater than the item on the right.
!=	A logical condition to indicate an item on the left is not equal to an item on the right.
&&	Logical "AND"
||	Logical "OR"
!	Logical "NOT"
+ - * /	Arithmetic operators
;	Separator of expressions

Readers are strongly recommended to refer to the subclauses for the attribute definitions, the local functions and the FDL-PDU definitions to understand protocol machines. It is assumed that readers have sufficient knowledge of these definitions and they are used without further explanations.

Further constructs as defined in C language notation (ISO/IEC 9899) can be used to describe conditions and actions.

4 Application Layer protocol specification

4.1 Operating principle

EtherCAT is a Real Time Ethernet technology that aims to maximize the utilization of the full duplex Ethernet bandwidth. Medium access control employs the master/slave principle, where the master node (typically the control system) sends the Ethernet frames to the slave nodes, which extract data from and insert data into these frames.

From an Ethernet point of view, a EtherCAT segment is a single Ethernet device, which receives and sends standard ISO/IEC 8802-3 Ethernet frames. However, this Ethernet device is not limited to a single Ethernet controller with a downstream microprocessor, but may consist of a large number of slave devices. These process the incoming frames directly and extract the relevant user data or insert data and transfer the frame to the next slave device. The last slave device within the segment sends the fully processed frame back, so that it is returned by the first slave device to the master as response frame.

This procedure utilizes the full duplex mode of Ethernet: both communication directions are operated independently. Direct communication without a switch between a master device and a EtherCAT segment consisting of one or several slave devices may be established.

Industrial communication systems have to meet different requirements in terms of the data transmission characteristics. Parameter data is transferred acyclically and in large quantities, whereby the timing requirements are relatively non-critical, and the transmission is usually triggered by the control system. Diagnostic data is also transferred acyclically and event-driven, but the timing requirements are more demanding, and the transmission is usually triggered by a peripheral device.

Process data, on the other hand, is typically transferred cyclically with different cycle times. The timing requirements are most stringent for process data communication. EtherCAT supports a variety of services and protocols to meet these differing requirements.

4.2 Node reference model

4.2.1 Mapping onto OSI basic reference model

EtherCAT is described using the principles, methodology and model of ISO/IEC 7498 Information processing systems — Open Systems Interconnection — Basic Reference Model (OSI). The OSI model provides a layered approach to communications standards, whereby the layers can be developed and modified independently. The EtherCAT specification defines functionality from top to bottom of a full OSI stack, and some functions for the users of the stack. Functions of the intermediate OSI layers, layers 3 – 6, are consolidated into either the EtherCAT Data Link layer or the EtherCAT Application layer. Likewise, features common to users of the Fieldbus Application Layer may be provided by the EtherCAT Application layer to simplify user operation, as noted in Figure 3.

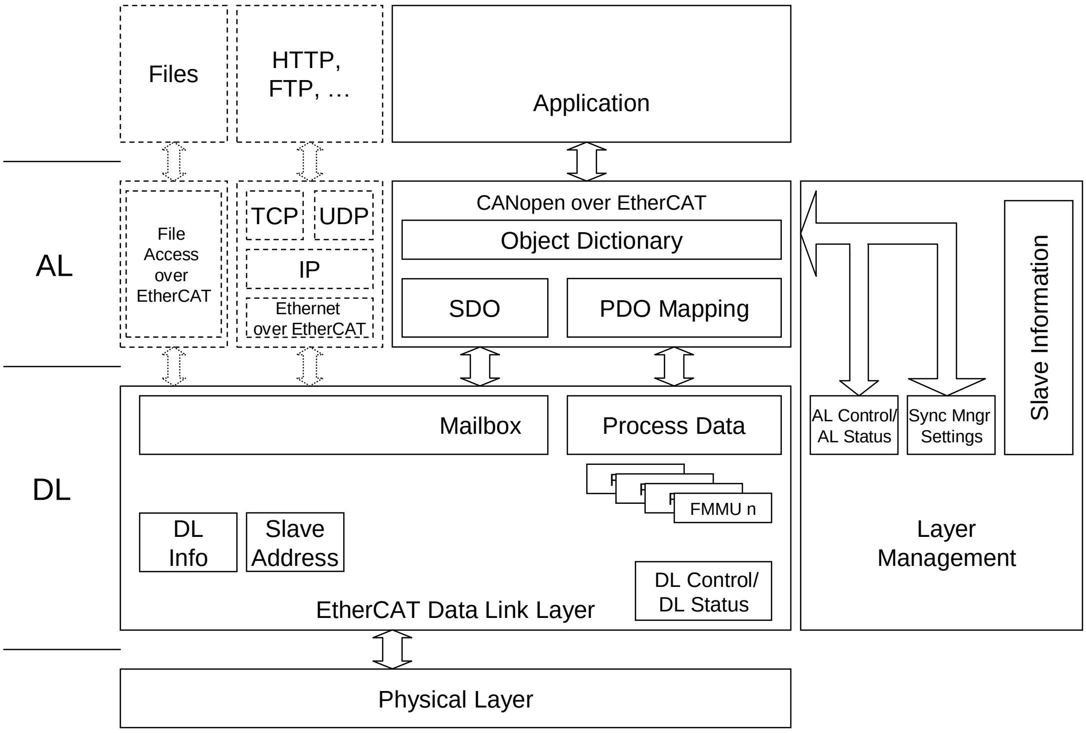

Figure 3 **– Slave Node Reference Model**

4.2.2 Data Link Layer features

The data link layer provides basic time critical support for data communications among devices. The term "time-critical" is used to describe applications having a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

The data link layer has the task to compute, compare and generate the frame check sequence and provide communications by extracting data from and/or including data into the Ethernet frame. This is done depending on the data link layer parameters which are stored at pre-defined memory locations. The application data is made available to the application layer in physical memory, either in a mailbox configuration or within the process data section.

Additionally, some data structures in the Data Link layer will be used to allow a coordination of the interaction between master and slave such as AL Control, Status and Event and Sync manager settings.

4.2.3 Application Layer structure

The Application Layer consists of the elements

- A real time entity (mandatory)
- An entity that deals with TCP/UDP/IP and related protocols (optional)
- A file access utility (optional)
- A Management unit (mandatory)

The Application Layer uses the services provided by the EtherCAT Data Link Layer to convey the Application Layer service data.

5 FAL syntax description

5.1 Coding principles

Application layer uses DL objects as defined in ETG.1000.4.

5.2 Data types and encoding rules

5.2.1 General description of data types and encoding rules

The format of this data and its meaning have to be known by the producer and consumer(s) to be able to exchange meaningful data. This specification models this by the concept of data types.

The encoding rules define the representation of values of data types and the transfer syntax for the representations. Values are represented as bit sequences. Bit sequences are transferred in sequences of octets (bytes). For numerical data types the encoding is little endian style as shown in Table 6.

The data types and encoding rules shall be valid for the DL services and protocols as well as for the AL services and protocols specified. The encoding rules for the Ethernet frame are specified in ISO/IEC 8802-3. The DLSDU of Ethernet is an octet string. The transmission order within octets depends upon MAC and PhL encoding rules.

5.2.2 Encoding of a Boolean value

a) The encoding of a Boolean value shall be primitive. The ContentsOctets shall consist of a single octet.

b) If the Boolean value is FALSE, the ContentsOctets shall be 0 (zero). If the Boolean value is TRUE, the ContentsOctets shall be 0xff.

5.2.3 Encoding of a Time Of Day with and without date indication value

a) The encoding of a Time Of Day with and without date indication value shall be primitive.

b) The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 4:

bits	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
octets 1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	number of milliseconds since midnight
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	number of days since 01.01.84 only with date indication
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	msb								

Figure 4 – Encoding of Time Of Day value

5.2.4 Encoding of a Time Difference with and without date indication value

a) The encoding of a Time Difference with and without date indication value shall be primitive.

b) The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 5:

bits	7	6	5	4	3	2	1	0	
octets									
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	milliseconds
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	days only with date indication
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	msb								

Figure 5 – Encoding of Time Difference value

5.2.5 Transfer syntax for bit sequences

For transmission a bit sequence is reordered into a sequence of octets. Hexadecimal notation is used for octets as specified in ISO/IEC 9899. Let $b = b_0...b_{n-1}$ be a bit sequence. Denote k a non-negative integer such that $8(k - 1) < n \leq 8k$. Then b is transferred in k octets assembled as shown in Table 6. The bits b_i, $i \geq n$ of the highest numbered octet are do not care bits.

Octet 1 is transmitted first and octet k is transmitted last. Hence the bit sequence is transferred as follows across the network (transmission order within an octet is determined by ISO/IEC 8802-3):

$b_7, b_6, ..., b_0, b_{15}, ..., b_8, ...$

Table 6 – Transfer Syntax for bit sequences

octet number	1.	2.	k.
	b_7 .. b_0	b_{15} .. b_8	b_{8k-1} .. b_{8k-8}

EXAMPLE

Bit 9	...	Bit 0
10b	0001b	1100b
0x2	0x1	0xC
		= 0x21C

The bit sequence b = b_0 .. b_9 = 0011 1000 01_b represents an Unsigned10 with the value 0x21C and is transferred in two octets: First 0x1C and then 0x02.

5.2.6 Encoding of a Unsigned Integer value

Data of basic data type Unsignedn has values in the non-negative integers. The value range is 0, ..., 2^n-1. The data is represented as bit sequences of length *n*. The bit sequence

$$b = b_0 \ldots b_{n-1}$$

is assigned the value

$$\text{Unsignedn}(b) = b_{n-1}\times2^{n-1}+ \ldots + b_1\times2^1 + b_0\times2^0$$

The bit sequence starts on the left with the least significant byte (Octet).

NOTE: Example: The value 266 = 0x10A with data type Unsigned16 is transferred in two octets, first 0x0A and then 0x01.

The Unsignedn data types are transferred as specified in Table 7. Unsigned data types as Unsigned1 to Unsigned7 and Unsigned 9 to Unsigned15 will be used too. In this case the next element will start at the first free bit position as denoted in 3.6.3.

Table 7 – Transfer syntax for data type Unsignedn

octet number	1.	2.	3.	4.	5.	6.	7.	8.
Unsigned8	$b_7..b_0$							
Unsigned16	$b_7..b_0$	$b_{15}..b_8$						
Unsigned32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Unsigned64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

BYTE is used as UNSIGNED8, WORD is used as UNSIGNED16.

5.2.7 Encoding of a Signed Integer value

Data of basic data type Integern has values in the integers. The value range is from -2^{n-1} to 2^{n-1}-1. The data is represented as bit sequences of length *n*. The bit sequence

$$b = b_0 \,..\, b_{n-1}$$

is assigned the value

$$\text{Integern}(b) = b_{n-2}\times2^{n-2} + \ldots + b_1\times2^1 + b_0\times2^0 \quad \text{if } b_{n-1} = 0$$

and, performing two's complement arithmetic,

$$\text{Integern}(b) = -\ \text{Integern}(\hat{}b) - 1 \qquad \text{if } b_{n-1} = 1$$

Note that the bit sequence starts on the left with the least significant bit.

Example: The value –266 = 0xFEF6 with data type Integer16 is transferred in two octets, first 0xF6 and then 0xFE.

The Integern data types are transferred as specified in Table 8. Integer data types as Integer1 to Integer7 and Integer9 to Integer15 will be used too. In this case the next element will start at the first free bit position as denoted in 3.6.3.

Table 8 – Transfer syntax for data type Integern

octet number	1.	2.	3.	4.	5.	6.	7.	8.
Integer8	$b_7..b_0$							
Integer16	$b_7..b_0$	$b_{15}..b_8$						
Integer32	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$				
Integer64	$b_7..b_0$	$b_{15}..b_8$	$b_{23}..b_{16}$	$b_{31}..b_{24}$	$b_{39}..b_{32}$	$b_{47}..b_{40}$	$b_{55}..b_{48}$	$b_{63}..b_{56}$

5.2.8 Encoding of a Floating Point value

```
Float32 ::= OCTET STRING SIZE (4)          -- IEC 60559 Single precision

Float64 ::= OCTET STRING SIZE (8)          -- IEC 60559 Double precision
```

5.2.9 Encoding of a Visible String value

a) The encoding of a variable length VisibleString value shall be primitive.

b) There is no Length field and no termination symbol; the length is encoded implicitly.

c) The ContentsOctets shall be a sequence of octets. The leftmost string element is encoded in the first octet, followed by the second octet, followed by each octet in turn up to and including the last octet as rightmost of the ContentsOctets.

5.2.10 Encoding of a Unicode String value

a) The encoding of a variable length UnicodeString value shall be primitive.

b) There is no Length field; the length is encoded implicitly.

c) The ContentsOctets shall be a sequence of unsigned integer. The leftmost string element is encoded in the first unsigned integer, followed by the second unsigned integer, followed by each unsigned integer in turn up to and including the last unsigned integer as rightmost of the ContentsOctets.

5.2.11 Encoding of an Octet String value

a) The encoding of a variable length OctetString value shall be primitive.

b) There is no Length field; the length is encoded implicitly.

c) The ContentsOctets shall be a sequence of octets. The leftmost string element is encoded in the first octet, followed by second octet, followed by each octet in turn up to and including the last octet as rightmost of the ContentsOctets.

5.2.12 Encoding of GUID

Data of basic data type GUID (Globally Unique Identifier) is a unique reference number used as an identifier. The value of a GUID is stored as a 128-bit integer.

5.3 AR coding

5.3.1 AL Control Request (Indication)

The attribute types of AL Control Request are described in Figure 6.

```
typedef struct
{
  unsigned          State:            4;
  unsigned          Acknowledge:      1;
  unsigned          IdRequest:        1;
  unsigned          Reserved:         2;
  unsigned          ApplSpecific:     8;
} TALCONTROL;
```

Figure 6 – AL Control Request structure

The AL Control Request is mapped to a DL write service to the DL-user control register object R1 and R2 as specified in ETG.1000.4. The AL Control Request coding is specified in Table 9.

Table 9 – AL Control Description

Parameter	DL-user Register	Data Type	Value
State	R1	Unsigned4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Acknowledge	R1	Unsigned1	0: Parameter Change of the AL Status Register will be unchanged 1: Parameter Change of the AL Status Register will be reset
IdRequest	R1	Unsigned1	0: ID is not requested 1: Request of ID
Reserved	R1	Unsigned2	Shall be zero
Reserved	R2	Unsigned8	

5.3.2 AL Control Response (Confirmation)

The AL Control Response is mapped to a DL read service to the DL-user status register object and register R4, R5 and R6 as specified in ETG.1000.4. The attribute types of AL Control Response are described in Figure 7.

```
typedef struct
{
  unsigned          State:           4;
  unsigned          Change:          1;
  unsigned          IdLoaded:        1;
  unsigned          Reserved1:       2;
  unsigned          ApplSpecific:    8;
  unsigned          Reserved2:      16;
  unsigned          AlStatusCode:   16;
} TALSTATUSE;
```

Figure 7 – AL Control Response structure

The AL Control Response coding is specified in Table 10.

Table 10 – AL Control Response

Parameter	DL-user Register	Data Type	Value
State	R3	Unsigned4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Change	R3	Unsigned1	0: State transition successful 1: State transition not successful
IdLoaded	R3	Unsigned1	0: Value in R6 does not represent ID 1: Value in R6 represents ID
Reserved	R3	Unsigned2	
Reserved	R4	Unsigned8	
Reserved	R5	Unsigned16	
AL Status Code	R6	Unsigned16	see Table 11

Table 11 – AL Status Codes

Code	Description	Current state (or state change)	Resulting state
0x0000	No error	Any	Current state
0x0001	Unspecified error	Any	I + E, P + E, S + E
0x0002	No Memory	Any	I + E, P + E, S + E
0x0003	Invalid Device Setup	P -> S	P + E
0x0005	Reserved due to compatibility reasons		
0x0011	Invalid requested state change	I -> S, I -> O, P -> O O -> B, S -> B, P -> B	I + E, P + E, S + E
0x0012	Unknown requested state	Any	I + E, P + E, S + E
0x0013	Bootstrap not supported	I -> B	I + E
0x0014	No valid firmware	I -> P	I + E
0x0015	Invalid mailbox configuration	I -> B	I + E
0x0016	Invalid mailbox configuration	I -> P	I + E
0x0017	Invalid sync manager configuration	P -> S, S -> O	Current state + E
0x0018	No valid inputs available	S, O, S -> O,	S + E
0x0019	No valid outputs	O, S -> O	S + E
0x001A	Synchronization error	O, S -> O	S + E
0x001B	Sync manager watchdog	O, S	S + E
0x001C	Invalid Sync Manager Types	O, S, P -> S	S + E
0x001D	Invalid Output Configuration	O, S, P -> S	P + E
0x001E	Invalid Input Configuration	O, S, P -> S	P + E
0x001F	Invalid Watchdog Configuration	O, S, P -> S	P + E
0x0020	Slave needs cold start	Any	I + E, P + E, S + E
0x0021	Slave needs INIT	B, P, S, O	I + E, P + E, S + E
0x0022	Slave needs PREOP	S, O	S + E,
0x0023	Slave needs SAFEOP	O	S + E
0x0024	Invalid Input Mapping	P -> S	P + E
0x0025	Invalid Output Mapping	P -> S	P + E
0x0026	Inconsistent Settings	P -> S	P + E
0x0027	FreeRun not supported	P -> S	P + E
0x0028	SyncMode not supported	P -> S	P + E
0x0029	FreeRun needs 3Buffer Mode	P -> S	P + E
0x002A	Background Watchdog	S, O	P + E
0x002B	No Valid Inputs and Outputs	O, S -> O	S + E
0x002C	Fatal Sync Error	O	S + E
0x002D	No Sync Error	S -> O	S + E
0x0030	Invalid DC SYNC Configuration	O, S -> O, P -> S	P + E, S + E
0x0031	Invalid DC Latch Configuration	O, S -> O, P -> S	P + E, S + E
0x0032	PLL Error	O, S -> O	S + E
0x0033	DC Sync IO Error	O, S -> O	S + E

0x0034	DC Sync Timeout Error	O, S -> O	S + E
0x0035	DC Invalid Sync Cycle Time	P -> S	P + E
0x0036	DC Sync0 Cycle Time	P -> S	P + E
0x0037	DC Sync1 Cycle Time	P -> S	P + E
0x0041	MBX_AOE	B, P, S, O	I + E, P + E, S + E
0x0042	MBX_EOE	B, P, S, O	I + E, P + E, S + E
0x0043	MBX_COE	B, P, S, O	I + E, P + E, S + E
0x0044	MBX_FOE	B, P, S, O	I + E, P + E, S + E
0x0045	MBX_SOE	B, P, S, O	I + E, P + E, S + E
0x004F	MBX_VOE	B, P, S, O	I + E, P + E, S + E
0x0050	EEPROM no access	Any	I + E, P + E, S + E
0x0051	EEPROM Error	Any	I + E, P + E, S + E
0x0060	Slave restarted locally	Any	I
0x0061	Device Identification value updated	P	P + E
0x0062 …0x00EF	Reserved		
0x00F0	Application controller available	I	I + E
other codes < 0x8000	reserved		
0x8000 – 0xFFFF	Vendor specific		

5.3.3 AL State Changed

The AL State Changed is mapped to a DL read service to the AL Status and AL Status Code object. The attribute types of AL State Changed are described in Figure 8.

```
typedef struct
{
  unsigned          State:          4;
  unsigned          Change:         1;
  unsigned          Reserved1:      3;
  unsigned          ApplSpecific:   8;
  unsigned          Reserved2:     16;
  unsigned          AlStatusCode:  16;
} TALSTATUSE;
```

Figure 8 – AL State Changed structure

The AL State Changed coding is specified in Table 12.

Table 12 – AL State Changed

Parameter	DL-user Register	Data Type	Value
State	R3	Unsigned4	1: Init 2: Pre-Operational 3: Bootstrap 4: Safe-Operational 8: Operational
Change	R3	Unsigned1	Shall be one
Reserved	R3	Unsigned3	
Application Specific	R4	Unsigned8	
Reserved	R5	Unsigned16	

AL Status Code	R6	Unsigned16	see Table 11

5.3.4 AL AR Attributes

AL AR attributes can be accessed by DL read or write services or by local read write services.

The attribute types of PDI Control are described in Figure 9.

```
typedef struct
{
  unsigned      PDIType:                     8;
  unsigned      StrictALControl:          1;
  unsigned      Reserved:                 7;
} TPDICONTROL;
```

Figure 9 – PDI Control type description

The PDI Control coding is specified in Table 13. PDI Control will be loaded from SII at start-up.

Table 13 – PDI Control

Parameter	DL-user Register	Data Type	Access DL	Access local	Value/Description
PDI Type	R7	Unsigned8	R	R	Type specific (see ETG.1000.3 DL information parameter)
Strict AL Control	Copy	Unsigned1	R	R	0x00: AL Management will be done by an application Controller 0x01: AL Management will be emulated (AL status follows directly AL control)

The PDI Configuration coding is controller specific as shown in Table 14 and set by SII at start-up.

Table 14 – PDI Configuration

Parameter	DL-user Register	Data Type	Access DL	Access local	Value/Description
Application Specific	R8	unsigned8	R	R	

The attribute types of Sync Configuration are described in Figure 10.

```
typedef struct
{
  unsigned      SignalCondSync0:          2;
  unsigned      EnableSignalSync0:        1;
  unsigned      EnableInterruptSync0:     1;
  unsigned      SignalCondSync1:          2;
  unsigned      EnableSignalSync1:        1;
  unsigned      EnableInterruptSync1:     1;
} TSYNCCFG;
```

Figure 10 – Sync Configuration type description

The Sync Configuration coding is specified in Table 15. Sync Configuration will be loaded from SII at start-up.

Table 15 – Sync Configuration

Parameter	DL-user register	Data Type	Access DL	Access local	Value/Description
Signal Conditioning Sync0	R8	unsigned2	R	R	Controller specific
Enable Signal Sync0	R8	unsigned1	R	R	0x00: disable 0x01: enable
Enable Interrupt Sync0	R8	unsigned1	R	R	0x00: disable 0x01: enable
Signal Conditioning Sync1	R8	unsigned2	R	R	Controller specific
Enable Signal Sync1	R8	unsigned1	R	R	0x00: disable 0x01: enable
Enable Interrupt Sync1	R8	unsigned1	R	R	0x00: disable 0x01: enable

5.4 SII coding

The Slave Information Interface Area coding is specified in Table 16 and Table 17. Address means a word address (e.g. 0 is first word, 1 is second word).

Table 16 – Slave Information Interface Area

Parameter	Address	Data Type	Value/Description
PDI Control	0x0000	Unsigned16	Initialization value for PDI Control register (0x140-0x141)
PDI Configuration	0x0001	Unsigned16	Initialization value for PDI Configuration register (0x150-0x151)
SyncImpulseLen	0x0002	Unsigned16	Sync Impulse in multiples of 10 ns
PDI Configuration2	0x0003	Unsigned16	Initialization value for PDI Configuration register R8 most significant word (0x152-0x153)
Configured Station Alias	0x0004	Unsigned16	Alias Address
Reserved	0x0005	BYTE[4]	Reserved for future use
Checksum	0x0007	Unsigned16	low byte contains remainder of division of word 0 to word 6 as unsigned number divided by the polynomial x^8+x^2+x+1(initial value 0xFF)
Vendor ID	0x0008	Unsigned32	CAN-Object 0x1018, Subindex 1
Product Code	0x000A	Unsigned32	CAN-Object 0x1018, Subindex 2
Revision Number	0x000C	Unsigned32	CAN-Object 0x1018, Subindex 3
Serial Number	0x000E	Unsigned32	CAN-Object 0x1018, Subindex 4
Reserved	0x0010	BYTE[8]	Reserved for future use
Bootstrap Receive Mailbox Offset	0x0014	Unsigned16	Receive Mailbox Offset for Bootstrap state (master to slave)
Bootstrap Receive Mailbox Size	0x0015	Unsigned16	Receive Mailbox Size for Bootstrap state (master to slave) Standard Mailbox size and Bootstrap Mailbox can differ. A bigger Mailbox size in Bootstrap mode can be used for optimiziation
Bootstrap Send Mailbox Offset	0x0016	Unsigned16	Send Mailbox Offset for Bootstrap state (slave to master)
Bootstrap Send Mailbox Size	0x0017	Unsigned16	Send Mailbox Size for Bootstrap state (slave to master)

Parameter	Address	Data Type	Value/Description
			Standard Mailbox size and Bootstrap Mailbox can differ. A bigger Mailbox size in Bootstrap mode can be used for optimiziation
Standard Receive Mailbox Offset	0x0018	Unsigned16	Receive Mailbox Offset for Standard state (master to slave)
Standard Receive Mailbox Size	0x0019	Unsigned16	Receive Mailbox Size for Standard state (master to slave)
Standard Send Mailbox Offset	0x001A	Unsigned16	Send Mailbox Offset for Standard state (slave to master)
Standard Send Mailbox Size	0x001B	Unsigned16	Send Mailbox Size for Standard state (slave to master)
Mailbox Protocol	0x001C	Unsigned16	Mailbox Protocols Supported as defined in Table 18
Reserved	0x001D	BYTE[66]	Reserved for future use
Size	0x003E	Unsigned16	size of E2PROM in [KiBit] + 1 NOTE: KiBit means 1024 Bit. NOTE: size = 0 means a EEPROM size of 1 KiBit
Version	0x003F	Unsigned16	This Version is 1

Table 17 – Slave Information Interface Categories

Parameter	Address	Data Type	Value/Description
First Category Header	0x0040	Unsigned15	Category Type as defined in Table 19
	0x0040	Unsigned1	Vendor Specific
	0x0041	Unsigned16	Following Category Word Size x
First Category Data	0x0042	Category dependent	Category Data
Second Category Header	0x0042 + x	Unsigned15	Category Type as defined in Table 19
	0x0042 + x	Unsigned1	Vendor Specific
	0x0043 + x	Unsigned16	Following Category Word Size
Second Category Data	0x0044 + x	Category dependent	Category Data
…			Continuation of the scheme until last category

Table 18 – Mailbox Protocols Supported Types

Protocol	Value	Description
AoE	0x0001	ADS over EtherCAT (routing and parallel services)
EoE	0x0002	Ethernet over EtherCAT (tunnelling of Data Link services)
CoE	0x0004	CAN application protocol over EtherCAT (access to SDO)
FoE	0x0008	File Access over EtherCAT
SoE	0x0010	Servo Drive Profile over EtherCAT
VoE	0x0020	Vendor specific protocol over EtherCAT

Table 19 – Categories Types

Protocol	Value	Description
NOP	00	No info
Device specific	01 – 09	Device specific categories shall not be overwritten by Master or configuration tool. Might be used for calibration values)
STRINGS	10	String repository for other Categories structure of this category data see Table 20
DataTypes	20	Data Types for future use
General	30	General information structure of this category data see Table 21
FMMU	40	FMMUs to be used structure of this category data see Table 23
SyncM	41	Sync Manager Configuration structure of this category data see Table 24
TXPDO	50	TxPDO description structure of this category data see Table 25
RXPDO	51	RxPDO description structure of this category data see Table 25
DC	60	Distributed Clock for future use
Reserved	60-2047	Reserved
Vendor specific	0x0800-0x0FFF	Vendor specific categories
Reserved	0x1000-0xFFFF	Reserved
End	0xffff	

Table 20 – Structure Category String

Parameter	Byte Address	Data Type	Value/Description
nStrings	0x0000	Unsigned8	Number of Strings
str1_len	0x0001	Unsigned8	Length String1
str_1	0x0002	BYTE [str1_len]	String1 Data
str2_len	0x0002+str1_len	Unsigned8	Length String2
str_2	0x0003+str1_len	BYTE [str2_len]	String2 Data
….			
strn_len	0x000z	Unsigned8	Length Stringn
str_n	0x000z+1	BYTE [strn_len]	Stringn Data
PAD_Byte	0x000y	BYTE	Padding if Category length is odd

NOTE 1: String means VISIBLESTRING data type by default

NOTE 2: the first string is referenced by 1 and following, a string index of 0 refers to an empty string

Table 21 – Structure Category General

Parameter	Byte Address	Data Type	Value/Description
GroupIdx	0x0000	Unsigned8	Group Information (Vendor specific) - Index to STRINGS
ImgIdx	0x0001	Unsigned8	Image Name (Vendor specific) - Index to STRINGS
OrderIdx	0x0002	Unsigned8	Device Order Number (Vendor specific) - Index to STRINGS
NameIdx	0x0003	Unsigned8	Device Name Information (Vendor specific) - Index to STRINGS
Reserved	0x0004	Unsigned8	reserved
CoE Details	0x0005	Unsigned8	Bit 0: Enable SDO Bit 1: Enable SDO Info Bit 2: Enable PDO Assign Bit 3: Enable PDO Configuration Bit 4: Enable Upload at startup Bit 5: Enable SDO complete acces
FoE Details	0x0006	Unsigned8	Bit 0: Enable FoE
EoE Details	0x0007	Unsigned8	Bit 0: Enable EoE
SoEChannels	0x0008	Unsigned8	reserved
DS402Channels	0x0009	Unsigned8	reserved
SysmanClass	0x000a	Unsigned8	reserved
Flags	0x000b	Unsigned8	Bit 0: Enable SafeOp Bit 1: Enable notLRW Bit 2: MboxDataLinkLayer Bit 3,4: Selection of identification method as defined in Table 22
CurrentOnEBus	0x000c	Signed16	EBus Current Consumption in mA, negative Values means feeding in current feed in sets the available current value to the given value
GroupIdx	0x000e	Unsigned8	Index to Strings – duplicate for compatibility reasons
Reserved	0x000f	BYTE[1]	Reserved
Physical Port	0x0010	Unsigned16	Description of Physical Ports: 0x00: not use 0x01: MII 0x02: reserved 0x03: EBUS 0x04: Fast Hot Connect Each port is describe by 4 bits: 3:0: Port 0 7:4: Port 1 11:8: Port 2 15:9: Port3 NOTE: Fast Hot Connect means a Port with Ethernet Physical Layer and Autonegotiation off (100Mbps fullduplex)
Physical Memory Address	0x0012	Unsigned16	Element defines the ESC memory address where the Identification ID is saved if Identification Method = IdentPhyM
Reserved2	0x0014	BYTE[12]	reserved

Table 22 – Identification Methods

Parameter	Value	Description
IdentALSts	Bit3	ID selector mirrored in AL Statud Code
IdentPhyM	Bit4	ID selector value mirrored in specific physical memory as deonted by the parameter “Physical Memory Address”

Table 23 – Structure Category FMMU

Parameter	Byte Address	Data Type	Value/Description
FMMU0	0x0000	Unsigned8	0x00: FMMU0 not used 0x01: FMMU0 used for Outputs 0x02: FMMU0 used for Inputs 0x03: FMMU0 used for SyncM Status(Read Mailbox) 0xFF: FMMU0 not used
FMMU1	0x0001	Unsigned8	0x00: FMMU1 not used 0x01: FMMU1 used for Outputs 0x02: FMMU1 used for Inputs 0x03: FMMU1 used for SyncM Status(Read Mailbox) 0xFF: FMMU1 not used
	…		continued if more than 2 FMMU used

Table 24 – Structure Category SyncM for each Element

Parameter	Byte Address	Data Type	Value/Description
Physical Start Address	0x0000	WORD	Origin of Data (see Physical Start Address of SyncM)
Length	0x0002	WORD	
Control Register	0x0004	Unsigned8	Defines Mode of Operation (see Control Register of SyncM)
Status Register	0x0005	BYTE	don’t care
Enable Synch Manager	0x0006	Unsigned8	Enable SynchM Bit 0: enable Bit 1: fixed content (info for config tool –SyncMan has fixed content) Bit 2: virtual SyncManager (virtual SyncMan – no hardware resource used) Bit 3: opOnly (SyncMan should be enabled only in OP state) Bit 7:4: reserved
Sync Manager Type	0x0007	BYTE	SyncManager Type 0x00 = not used or unknown 0x01 = used for mailbox out 0x02 = used for mailbox in 0x03 = used for process data outputs 0x04 = used for process data inputs

NOTE Data from Offset 0x006 and 0x007 can not be used as register content for the corresponding ESC registers

Table 25 – Structure Category TXPDO and RXPDO for each PDO

Parameter	Address	Data Type	Value/Description
PDO Index	0x0000	Unsigned16	For RxPDO: 0x1600 to 17FF, For TxPDO: 0x1A00 to 1BFF
nEntry	0x0002	Unsigned8	Number of Entries
SyncM	0x0003	Unsigned8	Related Sync Manager
Synchronization	0x0004	Unsigned8	Reference to DC Synch
NameIdx	0x0005	Unsigned8	Name of the Object - Index to STRINGS
Flags	0x0006	WORD	for future use
Entry 1	0x0008	8 BYTES	repeated for each entry as defined in Table 26
…			
Entry nEntry	nEntry*8	8 BYTES	repeated for each entry as defined in Table 26

Table 26 – Structure PDO Entry

Parameter	Offset within entry structure	Data Type	Value/Description
Entry Index	0x0000	Unsigned16	Index of the entry
Subindex	0x0002	Unsigned8	Subindex
Entry Name Idx	0x0003	Unsigned8	Name of the Entry - Index to STRINGS
Data Type	0x0004	Unsigned8	Data Type of the entry (Index in CoE Object Dictionary)
BitLen	0x0005	Unsigned8	Data Length of the entry
Flags	0x0006	WORD	for future use

5.5 Isochronous PDI coding

The attribute types of Distributed Clock sync and latch are described in Figure 11.

```
typedef struct
{
  BYTE          Reserved1;
  unsigned      CyclicOperationEnable:   1;
  unsigned      SYNC0Activate:           1;
  unsigned      SYNC1Activate:           1;
  unsigned      Reserved2:               5;
  WORD          SYNCPulse;
  BYTE          Reserved5[10];
  unsigned      Interrupt1Status:        1;
  unsigned      Reserved2:               7;
  unsigned      Interrupt2Status:        1;
  unsigned      Reserved3:               7;
  DWORD         CyclicOperationStartTime;
  BYTE          Reserved4[12];
  DWORD         SYNC0CycleTime;
  DWORD         SYNC1CycleTime;
  unsigned      Latch0PosEdge:           1;
  unsigned      Latch0NegEdge:           1;
  unsigned      Reserved5:               14;
  unsigned      Latch1PosEdge:           1;
  unsigned      Latch1NegEdge:           1;
  unsigned      Reserved6:               14;
  BYTE          Reserved7[4];
  unsigned      Latch0PosEvnt:           1;
  unsigned      Latch0NegEvnt:           1;
  unsigned      Reserved8:               6;
  unsigned      Latch1PosEvnt:           1;
  unsigned      Latch1NegEvnt:           1;
  unsigned      Reserved9:               6;
  DWORD         Latch0PosEdgeValue;
  BYTE          Reserveda[4];
  DWORD         Latch0NegEdgeValue;
  BYTE          Reservedb[4];
  DWORD         Latch1PosEdgeValue;
  BYTE          Reservedc[4];
  DWORD         Latch1NegEdgeValue;
  BYTE          Reservedd[4];
} TDCISOCHRON;
```

Figure 11 – Distributed Clock sync and latch type description

The Distributed Clock sync parameter encoding is described in Table 27. The parameters are mapped to DL DC user parameter P1 to P6. The events SYNC0 and SYNC1 are mapped to the DL events

Table 27 – Distributed Clock sync parameter

Parameter	DC user parameter	Data Type	Access Type EtherCAT DL	Access Type PDI	Value/Description
Cyclic Operation Enable	P1	Unsigned1	RW	R	0: disabled 1: enabled
SYNC0 activate	P1	Unsigned1	RW	R	0: deactivated 1: SCNC0 pulse generated
SYNC1 activate	P1	Unsigned1	RW	R	0: deactivated 1: SCNC1 pulse generated
SYNC Pulse	P2	Unsigned16	R	R	Taken from SII
Interrupt 0 Status	P3	Unsigned1	R	R	0: not active 1: active
Reserved	P3	Unsigned7	R	R	
Interrupt 1 Status	P3	Unsigned1	R	R	0: not active 1: active
Reserved	P3	Unsigned7	R	R	
Cyclic Operation Start Time	P4	Unsigned32	RW	R	The interrupt generation will start when the lower 32 bits of the system time will reach this value (in ns)
SYNC0 Cycle Time	P5	DWORD	RW	R	Cycle time of SYNC0
SYNC1 Cycle Time	P6	DWORD	RW	R	Cycle time of SYNC1

The Distributed Clock latch data encoding is described in Table 28.

Table 28 – Distributed Clock latch data

Parameter	DC user parameter	Data Type	Access Type EtherCAT DL	Access Type PDI	Value/Description
Latch0 positive Edge	P7	Unsigned1	RW	R	0: continuous 1: single
Latch0 negative Edge	P7	Unsigned1	RW	R	0: continuous 1: single
Reserved	P7	Unsigned6	R	R	
Latch1 positive Edge	P7	Unsigned1	RW	R	0: continuous 1: single
Latch1 negative Edge	P7	Unsigned1	RW	R	0: continuous 1: single
Reserved	P7	Unsigned6	R	R	
Latch0 positive Event	P8	Unsigned1	RW	R	0: no Event 1: Event stored
Latch0 negative Event	P8	Unsigned1	RW	R	0: no Event 1: Event stored
Reserved	P8	Unsigned6	R	R	
Latch1 positive Event	P8	Unsigned1	RW	R	0: no Event 1: Event stored
Latch1 negative Event	P8	Unsigned1	RW	R	0: no Event 1: Event stored
Reserved	P8	Unsigned6	R	R	
Latch 0 positive Edge Value	P9	DWORD	R	R	Latch0 Value positive Event
Latch 0 negative Edge Value	P10	DWORD	R	R	Latch0 Value negative Event
Latch 1 positive Edge Value	P11	DWORD	R	R	Latch1 Value positive Event
Latch 1 negative Edge Value	P12	DWORD	R	R	Latch1 Value negative Event

5.6 CoE coding

5.6.1 PDU structure

The general attribute types of CoE are described in Figure 12.

```
typedef struct
{
  unsigned          NumberLo:         8;
  unsigned          NumberHi:         1;
  unsigned          Reserved:         3;
  unsigned          Service:          4;
} TCOEHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  BYTE              Data[MBX_DATA_SIZE-2];
} TCOEMBX;
```

Figure 12 – CoE general structure

The CoE coding is specified in Table 29.

Table 29 – CoE elements

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	Depending on the CoE service
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x01: Emergency 0x02: SDO Request 0x03: SDO Response 0x04: TxPDO 0x05: RxPDO 0x06: TxPDO remote request 0x07: RxPDO remote request 0x08: SDO Information

5.6.2 SDO

5.6.2.1 SDO Download Expedited

5.6.2.1.1 SDO Download Expedited Request

The attribute types of SDO Download Expedited Request are described in Figure 13.

```
typedef struct
{
  unsigned          SizeIndicator:    1;
  unsigned          TransferType:     1;
  unsigned          DataSetSize:      2;
  unsigned          CompleteAccess:   1;
  unsigned          Command:          3;
  BYTE              IndexLo;
  BYTE              IndexHi;
  BYTE              SubIndex;
} TINITSDOHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TINITSDOHEADER    SdoHeader;
  BYTE              Data[4];
} TINITSDODOWNLOADEXPREQMBX;
```

Figure 13 – SDO Download Expedited Request structure

The SDO Download Expedited Request coding is specified in Table 30.

Table 30 –SDO Download Expedited Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Size Indicator	Unsigned1	0x01: size of Data in Data Set Size specified
	Transfer Type	Unsigned1	0x01: Expedited transfer
	Data Set Size	Unsigned2	0x00: 4 Octet Data 0x01: 3 Octet Data 0x02: 2 Octet Data 0x03: 1 Octet Data
	Complete Access	Unsigned1	0x00: entry addressed with index and subindex will be downloaded 0x01: complete object will be downloaded, subindex shall be zero (subindex 0 included) or one (subindex 0 excluded)
	Command Specifier	Unsigned3	0x01: Download Request
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero or one if Complete Access = 0x01
	Data	BYTE[4]	Data of the Object Unused octets should be 0

5.6.2.1.2 SDO Download Expedited Response

The attribute types of SDO Download Expedited Response are described in Figure 14.

```
typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TINITSDOHEADER    SdoHeader;
} TINITSDODOWNLOADEXPRESMBX;
```

Figure 14 – SDO Download Expedited Response structure

The SDO Download Expedited Response coding is specified in Table 31.

Table 31 – SDO Download Expedited Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x03: SDO Response
SDO	Size Indicator	Unsigned1	0x00
	Transfer Type	Unsigned1	0x00
	Data Set Size	Unsigned2	0x00
	Complete Access	Unsigned1	Don’t care (recommend value is 0)
	Command Specifier	Unsigned3	0x03: Download Response
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero or one if Complete Access = 0x01
	reserved	DWORD	

5.6.2.2 SDO Download Normal

5.6.2.2.1 SDO Download Normal Request

The attribute types of SDO Download Normal Request are described in Figure 15.

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TCOEHEADER          CoeHeader;
  TINITSDOHEADER      SdoHeader;
  DWORD               CompleteSize;
  BYTE                Data[MBX_DATA_SIZE-10];
} TINITSDODOWNLOADNORMREQMBX;
```

Figure 15 – SDO Download Normal Request structure

The SDO Download Normal Request coding is specified in Table 32.

Table 32 – SDO Download Normal Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Size Indicator	Unsigned1	0x01
	Transfer Type	Unsigned1	0x00: Normal transfer
	Data Set Size	Unsigned2	0x00
	Complete Access	Unsigned1	0x00: entry addressed with index and subindex will be downloaded 0x01: complete object will be uploaded, subindex shall be zero (subindex 0 included) or one (subindex 0 excluded)
	Command Specifier	Unsigned3	0x01: Download Request
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero or one if Complete Access = 0x01
	Complete Size	DWORD	Complete Data Size of the Object
	Data	BYTE[n-10]	If ((Length-10) >= Complete Size): Data of the Object If ((Length-10) < Complete Size): First Data part of the Object, Download SDO Segment is following

5.6.2.2.2 SDO Download Normal Response

The attribute types and coding of SDO Download Normal Response are the same as of SDO Download Expedited Response (see 5.6.2.1.2).

5.6.2.3 Download SDO Segment

5.6.2.3.1 Download SDO Segment Request

The attribute types of Download SDO Segment Request are described in Figure 16.

```
typedef struct
{
  unsigned          MoreFollows:      1;
  unsigned          SegDataSize:      3;
  unsigned          Toggle:           1;
  unsigned          Command:          3;
} TSDOSEGHEADER;

typedef struct
```

```
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOSEGHEADER      SdoHeader;
  BYTE               Data[MBX_DATA_SIZE-3];
} TDOWNLOADSDOSEGREQMBX;
```

Figure 16 – Download SDO Segment Request structure

The Download SDO Segment Request coding is specified in Table 33.

Table 33 – Download SDO Segment Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	More Follows	Unsigned1	0x00: Download SDO Segment is following 0x01: last Download SDO Segment
	SegData Size	Unsigned3	Defines how much of the last 7 data Octets (which always has to be send) contain data (only applicable if Length is 0x0A, otherwise shall be 0x00): 0x00: 7 Octet Data 0x01: 6 Octet Data 0x02: 5 Octet Data 0x03: 4 Octet Data 0x04: 3 Octet Data 0x05: 2 Octet Data 0x06: 1 Octet Data 0x07: 0 Octet Data
	Toggle	Unsigned1	Shall toggle with every Download SDO Segment Request, starting with 0x00
	Command specifier	Unsigned3	0x00: Download Segment Request
	Data	BYTE[n-3]	Data part of the Object

5.6.2.3.2 Download SDO Segment Response

The attribute types of Download SDO Segment Response are described in Figure 17.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOSEGHEADER      SdoHeader;
} TDOWNLOADSDOSEGRESMBX;
```

Figure 17 – Download SDO Segment Response structure

The Download SDO Segment Response coding is specified in Table 34.

Table 34 – Download SDO Segment Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x03: SDO Response
SDO	Reserved	Unsigned4	0x00
	Toggle	Unsigned1	Shall be the same as for the corresponding Download SDO Segment Request
	Command Specifier	Unsigned3	0x01: Download Segment Response
	Reserved	BYTE[7]	

5.6.2.4 SDO Upload Expedited

5.6.2.4.1 SDO Upload Expedited Request

The attribute types of SDO Upload Expedited Request are described in Figure 18.

```
typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TINITSDOHEADER    SdoHeader;
} TINITSDOUPLOADEXPREQMBX;
```

Figure 18 – SDO Upload Expedited Request structure

The SDO Upload Expedited Request coding is specified in Table 35.

Table 35 – SDO Upload Expedited Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Reserved	Unsigned4	0x00
	Complete Access	Unsigned1	0x00: entry addressed with index and subindex will be uploaded 0x01: complete object will be uploaded, subindex shall be zero (subindex 0 included) or one (subindex 0 excluded)
	Command Specifier	Unsigned3	0x02: Upload Request
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero or one, if Complete Access = 0x01
	Reserved	DWORD	

5.6.2.4.2 SDO Upload Expedited Response

The attribute types of SDO Upload Expedited Response are described in Figure 19.

```
typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TINITSDOHEADER    SdoHeader;
  BYTE              Data[4];
} TINITSDOUPLOADEXPREQMBX;
```

Figure 19 – SDO Upload Expedited Response structure

The SDO Upload Expedited Response coding is specified in Table 36.

Table 36 – SDO Upload Expedited Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x03: SDO Response
SDO	Size Indicator	Unsigned1	0x01: size of Data in Data Set Size specified
	Transfer Type	Unsigned1	0x01: Expedited transfer
	Data Set Size	Unsigned2	0x00: 4 Octet Data 0x01: 3 Octet Data 0x02: 2 Octet Data 0x03: 1 Octet Data
	Complete Access	Unsigned1	0x00: entry addressed with index and subindex will be uploaded 0x01: complete object will be uploaded, subindex shall be zero (subindex 0 included) or one (subindex 0 excluded)
	Command Specifier	Unsigned3	0x02: Upload Response
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero or one, if Complete Access = 0x01
	Data	BYTE[4]	Data of the Object

5.6.2.5 SDO Upload Normal

5.6.2.5.1 SDO Upload Normal Request

The attribute types and coding of SDO Upload Normal Request are the same as of SDO Upload Expedited Request (see 5.6.2.4.1).

5.6.2.5.2 SDO Upload Normal Response

The attribute types of SDO Upload Normal Response are described in Figure 20.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TINITSDOHEADER     SdoHeader;
  DWORD              CompleteSize;
  BYTE               Data[MBX_DATA_SIZE-10];
} TINITSDOUPLOADNORMRESMBX;
```

Figure 20 – SDO Upload Normal Response structure

The SDO Upload Normal Response coding is specified in Table 37. If the number of octets of the Data parameter is equal or less than 4 the response as specified in 5.6.2.4.2 can be used.

Table 37 – SDO Upload Normal Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x03: SDO Response
SDO	Size Indicator	Unsigned1	0x01
	Transfer Type	Unsigned1	0x00: Normal transfer
	Data Set Size	Unsigned2	0x00
	Complete Access	Unsigned1	0x00: entry addressed with index and subindex will be uploaded 0x01: complete object will be uploaded, subindex shall be zero (subindex 0 included) or one (subindex 0 excluded)
	Command Specifier	Unsigned3	0x02: Upload Response
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object, shall be zero or one, if Complete Access = 0x01
	Complete Size	DWORD	Complete Data Size of the Object
	Data	BYTE[n-10]	If ((Length-10) >= Complete Size): Data of the Object If ((Length-10) < Complete Size): First Data part of the Object, Upload SDO Segment is following

5.6.2.6 Upload SDO Segment

5.6.2.6.1 Upload SDO Segment Request

The attribute types of Upload SDO Segment Request are described in Figure 21.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOSEGHEADER      SdoHeader;
} TUPLOADSDOSEGREQMBX;
```

Figure 21 – Upload SDO Segment Request structure

The Upload SDO Segment Request coding is specified in Table 38.

Table 38 – Upload SDO Segment Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Reserved	Unsigned4	0x00
	Toggle	Unsigned1	Shall toggle with every Upload SDO Segment Request, starting with 0x00
	Command Specifier	Unsigned3	0x03: Upload Segment Request
	Reserved	BYTE[7]	

5.6.2.6.2 Upload SDO Segment Response

The attribute types of Upload SDO Segment Response are described in Figure 22.

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TCOEHEADER          CoeHeader;
  TSDOSEGHEADER       SdoHeader;
  BYTE                Data[MBX_DATA_SIZE-3];
} TUPLOADSDOSEGRESMBX;
```

Figure 22 – Upload SDO Segment Response structure

The Upload SDO Segment Response coding is specified in Table 39.

Table 39 – Upload SDO Segment Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x03: SDO Response
SDO	More Follows	Unsigned1	0x00: Upload SDO Segment is following 0x01: last Upload SDO Segment
	SegData Size	Unsigned3	Defines how much of the last 7 data Octets (which always has to be send) contain data (only applicable if Length is 0x0A, otherwise shall be 0x00): 0x00: 7 Octet Data 0x01: 6 Octet Data 0x02: 5 Octet Data 0x03: 4 Octet Data 0x04: 3 Octet Data 0x05: 2 Octet Data 0x06: 1 Octet Data 0x07: 0 Octet Data
	Toggle	Unsigned1	Shall be the same as for the corresponding Upload SDO Segment Request
	Command specifier	Unsigned3	0x00: Upload Segment Response
	Data	BYTE[n-3]	Data part of the Object

5.6.2.7 Abort SDO Transfer

5.6.2.7.1 Abort SDO Transfer Request

The attribute types of Abort SDO Transfer Request are described in Figure 23.

```
typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TINITSDOHEADER    SdoHeader;
  DWORD             AbortCode;
} TABORTSDOTRANSFERREQMBX;
```

Figure 23 – Abort SDO Transfer Request structure

The Abort SDO Transfer Request coding is specified in Table 40.

Table 40 – Abort SDO Transfer Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x02: SDO Request
SDO	Size Indicator	Unsigned1	0x00
	Transfer Type	Unsigned1	0x00
	Data Set Size	Unsigned2	0x00
	Reserved	Unsigned1	0x00
	Command Specifier	Unsigned3	0x04: Abort Transfer Request
	Index	WORD	Index of the Object
	Subindex	BYTE	Subindex of the Object
	Abort Code	DWORD	Abort Code as specified in Table 41

5.6.2.7.2 SDO Abort Codes

The SDO Abort Codes are specified in Table 41.

Table 41 – SDO Abort Codes

Value	Meaning
0x05 03 00 00	Toggle bit not changed
0x05 04 00 00	SDO protocol timeout
0x05 04 00 01	Client/Server command specifier not valid or unknown
0x05 04 00 05	Out of memory
0x06 01 00 00	Unsupported access to an object
0x06 01 00 01	Attempt to read to a write only object
0x06 01 00 02	Attempt to write to a read only object
0x06 01 00 03	Subindex cannot be written, SI0 must be 0 for write access
0x06 01 00 04	SDO Complete access not supported for objects of variable length such as ENUM object types
0x06 01 00 05	Object length exceeds mailbox size
0x06 01 00 06	Object mapped to RxPDO, SDO Download blocked
0x06 02 00 00	The object does not exist in the object directory
0x06 04 00 41	The object can not be mapped into the PDO
0x06 04 00 42	The number and length of the objects to be mapped would exceed the PDO length
0x06 04 00 43	General parameter incompatibility reason
0x06 04 00 47	General internal incompatibility in the device
0x06 06 00 00	Access failed due to a hardware error
0x06 07 00 10	Data type does not match, length of service parameter does not match
0x06 07 00 12	Data type does not match, length of service parameter too high
0x06 07 00 13	Data type does not match, length of service parameter too low
0x06 09 00 11	Subindex does not exist
0x06 09 00 30	Value range of parameter exceeded (only for write access)
0x06 09 00 31	Value of parameter written too high
0x06 09 00 32	Value of parameter written too low
0x06 09 00 36	Maximum value is less than minimum value
0x08 00 00 00	General error
0x08 00 00 20	Data cannot be transferred or stored to the application NOTE: This is the general Abort Code in case no further detail on the reason can determined. It is recommended to use one of the more detailed Abort Codes (0x08000021, 0x08000022)
0x08 00 00 21	Data cannot be transferred or stored to the application because of local control NOTE: “local control” means an application specific reason. It does not mean the ESM-specific control
0x08 00 00 22	Data cannot be transferred or stored to the application because of the present device state NOTE: “device state” means the ESM state
0x08 00 00 23	Object dictionary dynamic generation fails or no object dictionary is present

5.6.3 SDO Information

5.6.3.1 General

Everything in the part "SDO Info Service Data" of the following services shall be handled as a block and be sent only once.

If the service have to be fragmented the Length field can be <= 0x0A for the last fragment

5.6.3.2 SDO Information Service

The attribute types of SDO Information Service are described in Figure 24.

```
typedef struct
{
  unsigned          OpCode:           7;
  unsigned          InComplete:       1;
  unsigned          Reserved:         8;
  WORD              FragmentsLeft;
} TSDOINFOHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TSDOINFOHEADER    SdoInfoHeader;
} TSDOINFOSERVICE;
```

Figure 24 – SDO Information Service structure

The SDO Information Service coding is specified in Table 42.

Table 42 – SDO Information Service

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n > 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x01: Get OD List Request 0x02: Get OD List Response 0x03: Get Object Description Request 0x04: Get Object Description Response 0x05: Get Entry Description Request 0x06: Get Entry Description Response 0x07: SDO Info Error Request
	Incomplete	Unsigned1	0x00: last SDO Information fragment 0x01: SDO Information fragments will follow
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Number of Fragments which will follow
SDO Info Service Data	Data	BYTE[n-6]	SDO Information Service Data

5.6.3.3 Get OD List

5.6.3.3.1 Get OD List Request

The attribute types of Get OD List Request are described in Figure 25.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOINFOHEADER     SdoInfoHeader;
  WORD               ListType;
} TGETODLISTREQ;
```

Figure 25 – Get OD List Request structure

The Get OD List Request coding is specified in Table 43.

Table 43 – Get OD List Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x08: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x01: Get OD List Request
	Incomplete	Unsigned1	Shall be zero
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Shall be zero
SDO Info Service Data	List Type	WORD	0x00: get number of objects in the 5 different lists 0x01: all objects of the object dictionary shall be delivered in the response 0x02: only objects which are mappable in a RxPDO shall be delivered in the response 0x03: only objects which are mappable in a TxPDO shall be delivered in the response 0x04: only objects which has to stored for a device replacement shall be delivered in the response 0x05: only objects which can be used as startup parameter shall be delivered in the response

5.6.3.3.2 Get OD List Response

The attribute types of Get OD List Response are described in Figure 26.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOINFOHEADER     SdoInfoHeader;
  WORD               ListType;
} TGETODLISTRES;
```

Figure 26 – Get OD List Response structure

The Get OD List Response coding is specified in Table 44.

Table 44 – Get OD List Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x08: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x02: Get OD List Response
	Incomplete	Unsigned1	0x00: last SDO Information fragment 0x01: SDO Information fragments will follow
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Number of Fragments which will follow SDO Info Header will be sent with every fragment, SDO Info Data will be sent fragmented
SDO Info Service Data	List Type	WORD	0x00: list of length shall be delivered in the response 0x01: all objects of the object dictionary shall be delivered in the response 0x02: only objects which are mappable in a RxPDO shall be delivered in the response 0x03: only objects which are mappable in a TxPDO shall be delivered in the response 0x04: only objects which has to stored for a device replacement shall be delivered in the response 0x05: only objects which can be used as startup parameter shall be delivered in the response
	Index List	WORD [(n-8)/2]	List of object indexes or 5 words with the length of the list types if list type is 0 Order of object indexes is free. It is recommended to sort in ascending order

5.6.3.4 OD List Segment

If the SDO Info Service Data of the Get OD List Response must be segmented the SDO Info Service Data are fragmented and the OD List Segment service shall transfer the fragments.

5.6.3.5 Get Object Description

5.6.3.5.1 Get Object Description Request

The attribute types of Get Object Description Request are described in Figure 27.

```
typedef struct
```

```
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOINFOHEADER     SdoInfoHeader;
  WORD               Index;
} TGETOBJDESCREQ;
```

Figure 27 – Get Object Description Request structure

The Get Object Description Request coding is specified in Table 45.

Table 45 – Get Object Description Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x08: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x03: Get Object Description Request
	Incomplete	Unsigned1	Shall be zero
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Shall be zero
SDO Info Service Data	Index	WORD	Index of the requested object description

5.6.3.5.2 Get Object Description Response

The attribute types of Get Object Description Response are described in Figure 28.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOINFOHEADER     SdoInfoHeader;
  WORD               Index;
  WORD               DataType;
  BYTE               MaxSubindex;
  BYTE               ObjCode;
} TGETOBJDESCRES;
```

Figure 28 – Get Object Description Response structure

The Get Object Description Response coding is specified in Table 46.

Table 46 – Get Object Description Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n > 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x04: Get Object Description Response
	Incomplete	Unsigned1	0x00: last SDO Information fragment
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Number of Fragments which will follow
SDO Info Service Data	Index	WORD	Index of the object description
	Data Type	WORD	Reference to data type list
	Max Subindex	BYTE	Maximum number of subindexes ob the object
	Object Code	BYTE	Object Code 7: Variable 8: Array 9: Record
	Name	char[n-12]	Name of the object

5.6.3.6 Get Entry Description

5.6.3.6.1 Get Entry Description Request

The attribute types of Get Entry Description Request are described in Figure 29.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOINFOHEADER     SdoInfoHeader;
  WORD               Index;
  BYTE               Subindex;
  BYTE               ValueInfo;
} TGETENTRYDESCREQ;
```

Figure 29 – Get Entry Description Request structure

The Get Entry Description Request coding is specified in Table 47.

Table 47 – Get Entry Description Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x05: Get Entry Description Request
	Incomplete	Unsigned1	Shall be zero
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Shall be zero
SDO Info Service Data	Index	WORD	Index of the requested object description
	Subindex	BYTE	Subindex of the requested object description
	ValueInfo	BYTE	The value info includes which elements shall be in the response: Bit 0: reserved Bit 1: reserved Bit 2: reserved Bit 3: unit type Bit 4: default value Bit 5: minimum value Bit 6: maximum value

5.6.3.6.2 Get Entry Description Response

The attribute types of Get Entry Description Response are described in Figure 30.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TCOEHEADER         CoeHeader;
  TSDOINFOHEADER     SdoInfoHeader;
  WORD               Index;
  BYTE               Subindex;
  BYTE               ValueInfo;
  WORD               DataType;
  WORD               BitLength;
  WORD               ObjAccess;
} TGETENTRYDESCRES;
```

Figure 30 – Get Entry Description Response structure

The Get Object Description Response coding is specified in Table 48.

Table 48 – Get Entry Description Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n >= 0x10: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x06: Get Entry Description Response
	Incomplete	Unsigned1	0x00: last SDO Information fragment 0x01: SDO Information fragments will follow
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Number of Fragments which will follow
SDO Info Service Data	Index	WORD	Index of the requested object description
	Subindex	BYTE	Subindex of the requested object description
	ValueInfo	BYTE	The value info includes which elements are in the response: Bit 0: reserved Bit 1: reserved Bit 2: reserved Bit 3: unit type Bit 4: default value Bit 5: minimum value Bit 6: maximum value
	Data Type	WORD	Index of the data type of the object
	Bit Length	WORD	Bit length of the object If the length is = 0xFFFF: the length of the object is greater than 64 kBit or for objects with variable length.. To get the length of the object this object has to be uploaded
	Object Access	WORD	Bit 0: read access in Pre-Operational state Bit 1: read access in Safe-Operational state Bit 2: read access in Operational state Bit 3: write access in Pre-Operational state Bit 4: write access in Safe-Operational state Bit 5: write access in Operational state Bit 6: object is mappable in a RxPDO Bit 7: object is mappable in a TxPDO Bit 8: object can be used for backup Bit 9: object can be used for settings Bit 10-15: reserved

Frame part	Data Field	Data Type	Value/Description
	Data	BYTE[n-16]	If the unit type is included in the response, the unit type of the object is following (DWORD) If the default value is included in the response, the default value of the object entry is following (same data type as the object value) If the minimum value is included in the response, the minimum value of the object entry is following (same data type as the object value) If the maximum value is included in the response, the maximum value of the object entry is following (same data type as the object value) If the Length is greater than the described response parameter, the description is following (array of char)

5.6.3.7 Entry Description Segment

The attribute types and coding of Entry Description Segment are the same as of Get Entry Description Response.

5.6.3.8 SDO Info Error

The attribute types of SDO Info Error Request are described in Figure 31.

```
typedef struct
{
  TMBXHEADER        MbxHeader;
  TCOEHEADER        CoeHeader;
  TSDOINFOHEADER    SdoInfoHeader;
  DWORD             AbortCode;
} TABORTSDOTRANSFERREQMBX;
```

Figure 31 – SDO Info Error Request structure

The SDO Info Error Request coding is specified in Table 49.

Table 49 – SDO Info Error Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x08: SDO Information
SDO Info Header	Opcode	Unsigned7	0x07: SDO Info Error Request
	Incomplete	Unsigned1	Shall be zero
	Reserved	Unsigned8	0x00
	Fragments Left	WORD	Shall be zero
	Abort Code	DWORD	Abort Code as specified in Table 41

5.6.4 Emergency

5.6.4.1 Emergency Request

The Emergency Request coding is specified in Table 50.

Table 50 – Emergency Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n = 0x0A: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	0x00
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x01: Emergency
Emergency	Error Code	WORD	Error Code
	Error Register	BYTE	Error Register
	Data	BYTE[5]	Error Code 0000-9FFF: Manufacturer Specific Error Field Error Code A000-EFFF: Diagnostic Data Error Code F000-FFFF: Manufacturer Specific Error Field
	Reserved	BYTE[n-10]	

5.6.4.2 Emergency Error Codes

The Emergency Error Codes are specified in Table 51.

Table 51 – Emergency Error Codes

Error Code (hex)	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Software
62xx	User Software
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
82xx	Protocol Error
8210	PDO not processed due to length error
8220	PDO length exceeded
90xx	External Error
A0xx	ESM Transition Error
F0xx	Additional Functions
FFxx	Device specific

5.6.4.3 ESM Transition Error

5.6.4.3.1 Error Code

The ESM Transition Error Codes are specified in Table 52.

Table 52 – Error Code

Error Code (hex)	Meaning
A000	Transition PRE-OPERATIONAL to SAFE-OPERATIONAL not successful
A001	Transition SAFE-OPERATIONAL to OPERATIONAL not successful

5.6.4.3.2 Diagnostic Data

The ESM Transition Diagnostic Data structure is specified in Table 53.

Table 53 – Diagnostic Data

Data[0]	Data[1..4]	Meaning
0x00 + channel*4	Sync Manager Length Error	Length of the Sync Manager channel does not match
0x01 + channel*4	Sync Manager Address Error	Physical Start Address of the Sync Manager channel does not match
0x02 + channel*4	Sync Manager Settings Error	Settings of the Sync Manager channel are not matching

5.6.4.3.3 Sync Manager Length Error

The Sync Manager Length Error coding is specified in Table 54.

Table 54 – Sync Manager Length Error

Data[1..4]	Data Type	Value/Description
Minimum Length	WORD	Minimum value for the parameter Length of the Sync Manager channel
Maximum Length	WORD	Maximum value for the parameter Length of the Sync Manager channel

5.6.4.3.4 Sync Manager Address Error

The Sync Manager Address Error coding is specified in Table 55.

Table 55 – Sync Manager Address Error

Data[1..4]	Data Type	Value/Description
Minimum Address	WORD	Minimum value for the parameter Physical Start Address of the Sync Manager channel
Maximum Address	WORD	Maximum value for the parameter Physical Start Address of the Sync Manager channel

5.6.4.3.5 Sync Manager Settings Error

The Sync Manager Settings Error coding is specified in Table 56.

Table 56 – Sync Manager Settings Error

Data[1..4]	Data Type	Value/Description
Expected Buffer Type	Unsigned2	Expected value for the parameter Buffer Type of the Sync Manager channel
Expected Direction	Unsigned2	Expected value for the parameter Direction of the Sync Manager channel
Reserved	Unsigned1	0x00 (Reserved for future)
Expected AL Event Enable	Unsigned1	Expected value for the parameter AL Event Enable of the Sync Manager channel
Reserved	Unsigned10	0x00 (Reserved for future)
Expected Channel Enable	Unsigned1	Expected value for the parameter Channel Enable of the Sync Manager channel
Reserved	Unsigned15	0x00 (Reserved for future)

5.6.5 Process Data

5.6.5.1 RxPDO

The protocol of the RxPDO Transmission via mailbox is specified in Table 57.

Table 57 – RxPDO Transmission via mailbox

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n > 0x02: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	Related RxPDO-Number
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x05: RxPDO
PDO	Data	BYTE[n-2]	Process Output Data

5.6.5.2 TxPDO

The TxPDO Transmission via mailbox coding is specified in Table 58.

Table 58 – TxPDO Transmission via mailbox

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	n > 0x02: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	Related TxPDO-Number
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x04: TxPDO
PDO	Data	BYTE[n-2]	Process Input Data

5.6.5.3 RxPDO Remote Transmission Request

The RxPDO Remote Transmission Request coding is specified in Table 59.

Table 59 – RxPDO Remote Transmission Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x02: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	Related RxPDO-Number
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x07: RxPDO Remote Transmission Request

5.6.5.4 TxPDO Remote Transmission Request

The TxPDO Remote Transmission Request coding is specified in Table 60.

Table 60 – TxPDO Remote Transmission Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x02: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x03: CoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
CoE Header	Number	Unsigned9	Related TxPDO-Number
	Reserved	Unsigned3	0x00
	Service	Unsigned4	0x06: TxPDO Remote Transmission Request

5.6.6 Command

The Command object structure is specified in Table 61. Each command object shall have the data type 0x0025. The structure can be used by any object declared as command object.

Table 61 – Command object structure

Subindex	Description	Data Type	Value
0	Number of entries	UNSIGNED8	3
1	Command	OCTET_STRING	Byte 0-n: Request Data A write access to the command data will execute the command
2	Status	UNSIGNED8	0: last command completed, no errors, no reply 1: last command completed, no errors, reply there 2: last command completed, error, no reply 3: last command completed, error, reply there 4-99: reserved for future use 100-200: indicates how of the command has been executed (in %, 100 = 0 %, 200 = 100 %) 201-254: reserved for future use 255: command is executing (if the percentage display is not supported)
3	Reply	OCTET-STRING	Byte 0-n: Response Data

5.6.7 Object Dictionary

5.6.7.1 Object Dictionary structure

The Object Dictionary is structured as noted in Table 62.

Table 62 – Object Dictionary Structure

Index (hex)	Object Dictionary Area
0x0000-0x0FFF	Data Type Area
0x1000-0x1FFF	CoE Communication Area
0x2000-0x5FFF	Manufacturer Specific Area
0x6000-0xFFFF	Profile Area

5.6.7.2 Object Code Definitions

The Object Code Definition entries are structured as noted in Table 63.

Table 63 – Object Code Definitions

Object Code	Object Name
0002	DOMAIN
0005	DEFTYPE
0006	DEFSTRUCT
0007	VAR
0008	ARRAY
0009	RECORD

5.6.7.3 Data Type Area

The Basic Data Type Area is specified in Table 64.

Table 64 – Basic Data Type Area

Index (hex)	Object Type	Name
0001	DEFTYPE	BOOLEAN
0002	DEFTYPE	INTEGER8
0003	DEFTYPE	INTEGER16
0004	DEFTYPE	INTEGER32
0005	DEFTYPE	UNSIGNED8
0006	DEFTYPE	UNSIGNED16
0007	DEFTYPE	UNSIGNED32
0008	DEFTYPE	REAL32
0009	DEFTYPE	VISIBLE_STRING
000A	DEFTYPE	OCTET_STRING
000B	DEFTYPE	UNICODE_STRING
000C	DEFTYPE	TIME_OF_DAY
000D	DEFTYPE	TIME_DIFFERENCE
000E		Reserved
000F	DEFTYPE	DOMAIN
0010	DEFTYPE	INTEGER24
0011	DEFTYPE	REAL64
0012	DEFTYPE	INTEGER40
0013	DEFTYPE	INTEGER48
0014	DEFTYPE	INTEGER56
0015	DEFTYPE	INTEGER64
0016	DEFTYPE	UNSIGNED24
0017		Reserved
0018	DEFTYPE	UNSIGNED40
0019	DEFTYPE	UNSIGNED48
001A	DEFTYPE	UNSIGNED56
001B	DEFTYPE	UNSIGNED64
001C		Reserved
001D	DEFTYPE	GUID
001E	DEFTYPE	BYTE
001F-002C		Reserved
002D	DEFTYPE	BitARR8
002E	DEFTYPE	BITARR16
002F	DEFTYPE	BITARR32

The Extended Data Type Area is specified in Table 65.

Table 65 – Extended Data Type Area

Index (hex)	Object	Name
0020		Reserved
0021	DEFSTRUCT	PDO_MAPPING
0022		Reserved
0023	DEFSTRUCT	IDENTITY
0024		Reserved
0025	DEFSTRUCT	COMMAND_PAR
0026-0028		Reserved
0029	DEFSTRUCT	SYNC_PAR
002A-002F		Reserved
0030	DEFTYPE	BIT1
0031	DEFTYPE	BIT2
0032	DEFTYPE	BIT3
0033	DEFTYPE	BIT4
0034	DEFTYPE	BIT5
0035	DEFTYPE	BIT6
0036	DEFTYPE	BIT7
0037	DEFTYPE	BIT8
0038-003F		Reserved
0040-005F	DEFSTRUCT	Manufacturer Specific Complex Data Types
0060-007F	DEFTYPE	Device Profile 0 Specific Standard Data Types
0080-009F	DEFSTRUCT	Device Profile 0 Specific Complex Data Types
00A0-00BF	DEFTYPE	Device Profile 1 Specific Standard Data Types
00C0-00DF	DEFSTRUCT	Device Profile 1 Specific Complex Data Types
00E0-00FF	DEFTYPE	Device Profile 2 Specific Standard Data Types
0100-011F	DEFSTRUCT	Device Profile 2 Specific Complex Data Types
0120-013F	DEFTYPE	Device Profile 3 Specific Standard Data Types
0140-015F	DEFSTRUCT	Device Profile 3 Specific Complex Data Types
0160-017F	DEFTYPE	Device Profile 4 Specific Standard Data Types
0180-019F	DEFSTRUCT	Device Profile 4 Specific Complex Data Types
01A0-01BF	DEFTYPE	Device Profile 5 Specific Standard Data Types
01C0-01DF	DEFSTRUCT	Device Profile 5 Specific Complex Data Types
01E0-01FF	DEFTYPE	Device Profile 6 Specific Standard Data Types
0200-021F	DEFSTRUCT	Device Profile 6 Specific Complex Data Types
0220-023F	DEFTYPE	Device Profile 7 Specific Standard Data Types
0240-025F	DEFSTRUCT	Device Profile 7 Specific Complex Data Types
0260-07FF		Reserved

The Enumerated Data Type Area occupies index 0x800 to 0xFFF. Each item has a data type that specifies the number of bits occupied (e.g. BIT3 or UNSIGNED16) and a list of entries that specifies integer value (data type is unsigned32) and the enumeration visible string as shown in Table 66.

Table 66 – Enumeration Definition

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of entries	UNSIGNED8	O	R	No	number of enumeration values n
	Padding	UNSIGNED8				0 Padding to maintain even octets boundary for the following objects
1	Enum1	OCTET STRING	O	R	No	UNSIGNED32 as integer value VISIBLE STRING as enumeration string
	…					
1	Enumn	OCTET STRING	O	R	No	UNSIGNED32 as integer value VISIBLE STRING as enumeration string

5.6.7.4 CoE Communication Area

CoE Communication Object Dictionary Area consists of the elements described in Table 67.

Table 67 – CoE Communication Area

Index (hex)	Object type	Name	Type	M/O/C
1000	VAR	Device Type	UNSIGNED32	M
1001		Error Register	UNSIGNED8	O
1002		Reserved		
::::	::::	::::	::::	
1007		Reserved		
1008	VAR	Manufacturer Device Name	VisibleString	O
1009	VAR	Manufacturer Hardware Version	VisibleString	O
100A	VAR	Manufacturer Software Version	VisibleString	O
100B		Reserved		
::::	::::	::::	::::	
1017		Reserved		
1018	RECORD	Identity Object	Identity (0x23)	M
1019		Reserved		
::::	::::	::::	::::	
15FF		Reserved		
1600	RECORD	1st receive PDO Mapping	PDO Mapping (0x21)	C
1601	RECORD	2nd receive PDO Mapping	PDO Mapping	C
::::	::::	::::	::::	
17FF	RECORD	512th receive PDO Mapping	PDO Mapping	C
1800		Reserved		
::::	::::	::::	::::	
19FF		Reserved		
1A00	RECORD	1st transmit PDO Mapping	PDO Mapping (0x21)	C

1A01	RECORD	2nd transmit PDO Mapping	PDO Mapping	C
::::	::::	::::	::::	
1BFF	RECORD	512th transmit PDO Mapping	PDO Mapping	C
1C00	ARRAY	Sync Manager Communication Type	UNSIGNED8	M
1C01		Reserved		
::::	::::	::::	::::	
1C0F		Reserved		
1C10	ARRAY	Sync Manager 0 PDO Assignment	UNSIGNED16	C
1C11	ARRAY	Sync Manager 1 PDO Assignment	UNSIGNED16	C
1C12	ARRAY	Sync Manager 2 PDO Assignment	UNSIGNED16	C
1C13	ARRAY	Sync Manager 3 PDO Assignment	UNSIGNED16	C
1C14	ARRAY	Sync Manager 4 PDO Assignment	UNSIGNED16	C
::::	::::	::::	::::	
1C2F	ARRAY	Sync Manager 31 PDO Assignment	UNSIGNED16	C
1C30	RECORD	Sync Manager 0 Synchronization		O
::::	::::	::::	::::	
1C4F	RECORD	Sync Manager 31 Synchronization		O
1C50		Reserved		
::::	::::	::::	::::	
1FFF		Reserved		

5.6.7.4.1 Device Type

The Device Type object dictionary entry (index 0x1000) is specified in Table 68.

Table 68 – Device Type

Attribute	Value
Name	Device Type
Object Code	VAR
Data Type	UNSIGNED32
Category	Mandatory
Access	Ro
PDO Mapping	No
Value	Bit 0-15: used device profile, 0x0000 if no standardized device profile is used Bit 16-31: additional information depending on the used device profile

5.6.7.4.2 Error Register

The Error Register object dictionary entry (index 0x1001) is specified in Table 69.

Table 69 – Error Register

Attribute	Value
Name	Error Register
Object Code	VAR
Data Type	UNSIGNED8
Category	Optional
Access	Ro
PDO Mapping	Optional
Value	Bit 0: generic error Bit 1: current error Bit 2: voltage error Bit 3: temperature error Bit 4: communication error Bit 5: device profile specific error Bit 6: reserved Bit 7: manufacturer specific error

5.6.7.4.3 Manufacturer Device Name

The Manufacturer Device Name object dictionary entry (index 0x1008) is specified in Table 70.

Table 70 – Manufacturer Device Name

Attribute	Value
Name	Manufacturer Device Name
Object Code	VAR
Data Type	VISIBLE_STRING
Category	Optional
Access	Ro
PDO Mapping	No
Value	name of the device as non zero terminated string

5.6.7.4.4 Manufacturer Hardware Version

The Manufacturer Hardware Version object dictionary entry (index 0x1009) is specified in Table 71.

Table 71 – Manufacturer Hardware Version

Attribute	Value
Name	Manufacturer Hardware Version
Object Code	VAR
Data Type	VISIBLE_STRING
Category	Optional
Access	Ro
PDO Mapping	No
Value	hardware version of the device as non zero terminated string

5.6.7.4.5 Manufacturer Software Version

The Manufacturer Software Version object dictionary entry (index 0x100A) is specified in Table 72.

Table 72 – Manufacturer Software Version

Attribute	Value
Name	Manufacturer Software Version
Object Code	VAR
Data Type	VISIBLE_STRING
Category	Optional
Access	R
PDO Mapping	No
Value	Software version of the device as non zero terminated string

5.6.7.4.6 Identity Object

The Identity Object dictionary entry (index 0x1018) is specified in Table 73.

Table 73 – Identity Object

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of entries	UNSIGNED8	M	R	No	4
1	Vendor ID	UNSIGNED32	M	R	No	Assigned uniquely by ETG
2	Product Code	UNSIGNED32	M	R	No	Assigned uniquely by Vendor
3	Revision Number	UNSIGNED32	M	R	No	Assigned uniquely by Vendor
4	Serial Number	UNSIGNED32	M	R	No	Assigned uniquely for this device by Vendor 0 if there is no serial number given

5.6.7.4.7 Receive PDO Mapping

The Receive PDO Mapping object dictionary entry (index 0x1600 – 0x17FF) is specified in Table 74.

Table 74 – Receive PDO Mapping

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of objects in this PDO	UNSIGNED8	M	R/RW [a]	No	0 – 254 writeable if variable mapping is supported
1	First Output Object to be mapped	UNSIGNED32	C	R/RW [a]	No	Bit 0-7: length of the mapped objects in bits (for a gap in the PDO: shall have the bit length of the gap) Bit 8-15: subindex of the mapped object (0 in case of a gap in the PDO) Bit 16-31: index of the mapped object(for a gap in the PDO: shall be zero)
..						
n	Last Output Object to be mapped	UNSIGNED32	C	R/RW [a]	No	
[a] writeable if variable PDO mapping is supported.						

5.6.7.4.8 Transmit PDO Mapping

The Transmit PDO Mapping object dictionary entry (index 0x1A00 – 0x1BFF) is specified in Table 75.

Table 75 – Transmit PDO Mapping

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of objects in this PDO	UNSIGNED8	M	R/RW [a]	No	0 – 254 writeable if variable mapping is supported
1	First Output Object to be mapped	UNSIGNED32	C	R/RW [a]	No	Bit 0-7: length of the mapped objects in bits (for a gap in the PDO: shall have the bit length of the gap) Bit 8-15: subindex of the mapped object (0 in case of a gap in the PDO) Bit 16-31: index of the mapped object(for a gap in the PDO: shall be zero)
..						
n	Last Output Object to be mapped	UNSIGNED32	C	R/RW [a]	No	
[a] writeable if variable PDO mapping is supported.						

5.6.7.4.9 Sync Manager Communication Type

The Sync Manager Communication Type object dictionary entry (index 0x1C00) is specified in Table 76.

Table 76 – Sync Manager Communication Type

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of used Sync Manager channels	UNSIGNED8	M	R	No	4 – 32
1	Communication Type Sync Manager 0	UNSIGNED8	C	R	No	configurable [a] 0: unused 1: mailbox receive (master to slave) 2: mailbox send (slave to master) 3: process data output 4: process data input(slave to master)
2	Communication Type Sync Manager 1	UNSIGNED8	C	R	No	configurable [a] 0: unused 1: mailbox receive (master to slave) 2: mailbox send (slave to master) 3: process data output 4: process data input(slave to master)
3	Communication Type Sync Manager 2	UNSIGNED8	C	R	No	configurable [a] 0: unused 1: mailbox receive (master to slave) 2: mailbox send (slave to master) 3: process data output 4: process data input(slave to master)
4	Communication Type Sync Manager 3	UNSIGNED8	C	R	No	configurable [a] 0: unused 1: mailbox receive (master to slave) 2: mailbox send (slave to master) 3: process data output 4: process data input(slave to master)
5 – n	Communication Type Sync Manager 4 – (n-1)	UNSIGNED8	C	R	No	configurable [a] 0: unused 1: mailbox receive (master to slave) 2: mailbox send (slave to master) 3: process data output 4: process data input(slave to master)

[a] The Sync Manager communication type shall be used in the following way:
Communication Type Sync Manager 0: 1 mailbox receive
Communication Type Sync Manager 1: 2 mailbox send
Communication Type Sync Manager 2: 3 process data output (may be used for Inputs if not Outputs supported)
Communication Type Sync Manager 3: 4 process data input

If mailbox is not supported, it shall be used in the following way:
Communication Type Sync Manager 0: 3 process data output (may be used for Inputs if not Outputs supported)
Communication Type Sync Manager 1: 4 process data input

5.6.7.4.10 Sync Manager PDO Assignment

5.6.7.4.10.1 Sync Manager Channel

The Sync Manager Channel 0-31 object dictionary entry (index 0x1C10-0x1C2F) is specified in Table 77. If a Sync Manager Channel is used as a mailbox Sync Manager the corresponding Object shall not be available or Subindex 0 shall be 0.

Table 77 – Sync Manager Channel 0-31

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of assigned TxPDOs	UNSIGNED8	C	R/RW [a]	No	0 – 254
1 – n	PDO Mapping object index of assigned PDO	UNSIGNED16	C	R/RW [a]	No	Either 0x1600: RxPDO 1 0x1601: RxPDO 2 … 0x17FF: RxPDO 512 Or 0x1A00: TxPDO 1 0x1A01: TxPDO 2 … 0x1BFF: TxPDO 512

[a] writeable if variable PDO assign is supported.

5.6.7.4.11 Sync Manager Synchronization

The Sync Manager Synchronization object dictionary entry (index 0x1C30-0x1C4F) is specified in Table 78.

Table 78 – Sync Manager Synchronization

Sub-Index	Description	Data type	M/O/C	Access	PDO Mapping	Value
0	Number of Synchronization Parameters	UNSIGNED8	O	R	No	1 – 3
1	Synchronization type	UNSIGNED16	O	RW	No	• 0: not synchronized • 1: Synchron – synchronized with AL Event on this Sync Manager • 2: DC Sync0 – synchronized with AL Event Sync0 • 3: DC Sync1 – synchronized with AL Event Sync1 • 32: SyncSm0 – synchronized with AL Event of SM0 • 33: SyncSm1 – synchronized with AL Event of SM1 • … • 63: SyncSm31 – synchronized with AL Event of SM31
2	Cycle time	UNSIGNED32	O	RW	No	time between two events in ns
3	Shift time	UNSIGNED32	O	RW	No	time between related AL event and the associated action in ns

5.7 EoE coding

5.7.1 Initiate EoE

5.7.1.1 Initiate EoE Request

The attribute types of Initiate EoE Request are described in Figure 32.

```
typedef struct
{
  unsigned          FrameType:          4;
  unsigned          Port:               4;
  unsigned          LastFragment:       1;
  unsigned          TimeAppended:       1;
  unsigned          TimeRequested:      1;
  unsigned          Reserved:           5;
  unsigned          FragmentNumber:     6;
  unsigned          CompleteSize:       6;
  unsigned          FrameNumber:        4;
} TEOEHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TEOEHEADER        EoeHeader;
  BYTE              Data[MAX_EOE_DATA_SIZE];
} TINIEOEREQ;
```

Figure 32 – EoE general structure

The Initiate EoE Request coding is specified in Table 79.

Table 79 – Initiate EoE Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N = 0x24+ y*0x20: Length of the Mailbox Service Data (y = 0 to 0x2F)
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
EoE Header	FrameType	Unsigned4	0x00
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x00: at least one EoE Fragment service is following 0x01: complete Ethernet frame is in the Data part
	Time Appended	Unsigned1	0x00: no time stamp value will be appended after the EoE Data in the last fragment 0x01: time stamp value will be appended after the EoE Data in the last fragment
	Time Request	Unsigned1	0x00: no time stamp valueof the send time requested 0x01: time stamp valueof the send time requested
	Reserved	Unsigned5	
	Fragment Number	Unsigned6	0x00
	Complete Size	Unsigned6	in 32-octet blocks Trunc ((complete size in octet of the Ethernet frame + 31)/32)
	Frame Number	Unsigned4	Number of the Ethernet frame
	EoE Data	BYTE[N-4]	Ethernet frame (without Preamble, SFD, FCS) first portion (N-4) octets (4 Octets less if TimeStamp included)
(optional)	TimeStamp	Unsigned32	time of frame receipt, in ns starting at beginning of DA

5.7.1.2 Initiate EoE Response

The attribute types of Initiate EoE Response are described in Figure 33.

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TEOEHEADER          EoeHeader;
  TimeStamp           Unsigned32;
} TINIEOERES;
```

Figure 33 – EoE Timestamp structure

The Initiate EoE Response coding is specified in Table 80.

Table 80 – Initiate EoE Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N = 0x08: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
EoE Header	FrameType	Unsigned4	0x03
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x00
	Time Appended	Unsigned1	0x01: time stamp value will be appended
	Time Request	Unsigned1	0x00: no time stamp valueof the send time requested
	Reserved	Unsigned5	
	Fragment Number	Unsigned6	0x00
	Complete Size	Unsigned6	0x00
	Frame Number	Unsigned4	Number of the Ethernet frame
	TimeStamp	Unsigned32	time of frame send, in ns starting at beginning of DA

5.7.2 EoE Fragment Data

The attribute types of EoE Fragment Data are described in Figure 34.

```
typedef struct
{
  TMBXHEADER         MbxHeader;
  TEOEHEADER         EoeHeader;
  BYTE               Data[MAX_EOE_DATA_SIZE];
} TEOEFRAGREQ;
```

Figure 34 – EoE Fragment Data structure

The EoE Fragment Data coding is specified in Table 81.

Table 81 – EoE Fragment Data

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N > 0x04: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
EoE Header	FrameType	Unsigned4	0x00
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x00: at least one EoE Fragment service is following 0x01: last Data part of this Ethernet frame (including time stamp)
	Time Appended	Unsigned1	0x00: no time stamp value will be appended after the EoE Data in the last fragment 0x01: time stamp value will be appended after the EoE Data in the last fragment
	Time Request	Unsigned1	0x00: no time stamp valueof the send time requested 0x01: time stamp valueof the send time requested
	Reserved	Unsigned5	
	Fragment Number	Unsigned6	0x01-0x2F: fragment number of the Ethernet frame fragment
	Offset	Unsigned6	Offset in 32-octet blocks of the Ethernet frame fragment
	Frame Number	Unsigned4	Number of the Ethernet frame
	EoE Data	BYTE[N-4]	Ethernet frame (without Preamble, SFD, FCS) first portion (N-4) octets (4 Octets less if Timestamp included)
(optional)	TimeStamp	Unsigned32	time of frame receipt, in ns starting at beginning of DA

5.7.3 Data element for EoE

The EoE Data coding is specified in Table 82.

Table 82 – EoE Data

Frame part	Data Field	Data Type	Value/Description
Ethernet	Dest MAC	BYTE[6]	Destination MAC Address as specified in ISO/IEC 8802-3
	Src MAC	BYTE[6]	Source MAC Address as specified in ISO/IEC 8802-3
(optional)	VLAN Tag	BYTE[4]	0x81, 0x00 and two Octets Tag Control Information as specified in IEEE 802.1Q
	Ether Type	BYTE[2]	assigned by IEEE
User Frame	Data		User data (octet string) or EoE parameter
	Padding	BYTE[n]	shall be inserted if DL PDU is shorter than 64 octets as specified in ISO/IEC 8802-3

5.7.4 Set IP Parameter

5.7.4.1 Set IP Parameter Request

The attribute types of Set IP Parameter Request are described in Figure 35.

```
typedef struct
{
  TMBXHEADER          MbxHeader;
  TEOEHEADER          EoeHeader;
  BYTE                Data[MAX_EOE_DATA_SIZE];
} TEOEFRAGREQ;
```

Figure 35 – Set IP Parameter Request structure

The coding of Set IP Parameter Request is described in Table 83.

Table 83 – Set IP Parameter Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N > 0x08: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
EoE Header	FrameType	Unsigned4	0x02
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x01: last Data part
	Time Appended	Unsigned1	0x00: no time stamp value will be appended
	Time Request	Unsigned1	0x00: no time stamp value of the send time requested
	Reserved	Unsigned5	
	Fragment Number	Unsigned6	0x00
	Offset	Unsigned6	0x00
	Frame Number	Unsigned4	0x00
EoE Parameter	MAC included	Unsigned1	1, MAC address according to ISO/IEC 8802-3
	IP address included	Unsigned1	1, IP address according to IETF RFC 791
	Subnet Mask included	Unsigned1	1, Subnet mask according to IETF RFC 791
	Default Gateway included	Unsigned1	1, Default Gateway address according to IETF RFC 791
	DNS Server IP Address included	Unsigned1	1, IP address of DNS server according to IETF RFC 791
	DNS Name included	Unsigned1	1, DNS name according to IETF RFC 791
	reserved	Unsigned26	
	MAC	BYTE[6]	MAC address according to ISO/IEC 8802-3
	IP address	BYTE[4]	IP address according to IETF RFC 791
	Subnet Mask	BYTE[4]	Subnet mask according to IETF RFC 791 and IETF RFC 826
	Default Gateway	BYTE[4]	Default Gateway address according to IETF RFC 791
	DNS Server IP Address	BYTE[4]	IP address of DNS server according to IETF RFC 791
	DNS Name	char[32]	DNS name according to IETF RFC 791

5.7.4.2 Set IP Parameter Response

The attribute types of Set IP Parameter Response are described in Figure 36.

```
typedef struct
{
  unsigned          FrameType:          4;
  unsigned          Port:               4;
  unsigned          LastFragment:       1;
  unsigned          TimeAppended:       1;
  unsigned          TimeRequested:      1;
  unsigned          Reserved:           5;
  unsigned          Result:            16;
} TEOEPARAHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TEOEPARAHEADER    EoeHeader;
} TEOEFRAGREQ;
```

Figure 36 – Set IP Parameter Response structure

The coding of Set IP Parameter Response is described in Table 84.

Table 84 – Set IP Parameter Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N = 0x04: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
EoE Header	FrameType	Unsigned4	0x03
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x01: last Data part
	Time Appended	Unsigned1	0x00: no time stamp value will be appended
	Time Request	Unsigned1	0x00: no time stamp value of the send time requested
	Reserved	Unsigned5	
	Result	Unsigned16	see Table 85

Table 85 – EoE Result Parameter

result code	meaning
0x0000	Success
0x0001	Unspecified Error
0x0002	Unsupported Frame Type
0x0201	No IP Support
0x0202	DHCP not supported If the Master sends Set IP Parameter Request with IP Address “0.0.0.0” the slave should send an DCHP_Discover to get an IP Address. If the slave does not support DHCP it should response with Result “DHCP not supported”
0x0401	No Filter Support

5.7.5 Set Address Filter

5.7.5.1 Set MAC Filter Request

The attribute types of Set MACFilter Request are described in Figure 37.

```
typedef struct
{
  TMBXHEADER        MbxHeader;
  TEOEHEADER        EoeHeader;
  BYTE              Data[MAX_EOE_DATA_SIZE];
} TEOEFRAGREQ;
```

Figure 37 – Set MAC Filter Request structure

The coding of Set MAC Filter Request is described in Table 86.

Table 86 – Set Address Filter Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N > 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0
EoE Header	FrameType	Unsigned4	0x04
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x01: last Data part
	Time Appended	Unsigned1	0x00: no time stamp value will be appended
	Time Request	Unsigned1	0x00: no time stamp value of the send time requested
	Reserved	Unsigned5	
	Fragment Number	Unsigned6	0x00
	Offset	Unsigned6	0x00
	Frame Number	Unsigned4	0x00
EoE Parameter	MAC filter count	Unsigned4	Count of MAC address according to ISO/IEC 8802-3 which are accepted by Ethernet Ports on this slave
	MAC filter mask	Unsigned2	Count of MAC address masks according to ISO/IEC 8802-3 which are combined with filter by Ethernet Ports on this slave
	Reserved	Unsigned1	
	Inhibit Broadcast	Unsigned1	Filter Broadcast messages
	Reserved	Unsigned8	
(conditional)	List of MAC Address	List of BYTE[6]	MAC address according to ISO/IEC 8802-3
(conditional)	List of MAC Address Filter	List of BYTE[6]	MAC address according to ISO/IEC 8802-3 A set bit means that this address bit of Destination MAC Address is compared with the corresponding entry in the List of MAC Address.

5.7.5.2 Set Address Filter Response

The attribute types of Set Address Filter Response are described in Figure 38.

```
typedef struct
{
  unsigned          FrameType:          4;
  unsigned          Port:               4;
  unsigned          LastFragment:       1;
  unsigned          TimeAppended:       1;
  unsigned          TimeRequested:      1;
  unsigned          Reserved:           5;
  unsigned          Result:            16;
} TEOEPARAHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TEOEPARAHEADER    EoeHeader;
} TEOEFRAGREQ;
```

Figure 38 – Set Address Filter Response structure

The coding of Set Address Filter Response is described in Table 87.

Table 87 – Set Address Filter Response

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N = 0x08: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x02: EoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
EoE Header	FrameType	Unsigned4	0x05
	Port	Unsigned4	0x00: send to no specific port 0x01-0x0F: specific ports selected
	Last Fragment	Unsigned1	0x01: last Data part
	Time Appended	Unsigned1	0x00: no time stamp value will be appended
	Time Request	Unsigned1	0x00: no time stamp value of the send time requested
	Reserved	Unsigned5	
	Result	Unsigned16	see Table 85

5.8 FoE Coding

5.8.1 Read Request

The attribute types of Read Request are described in Figure 39.

```
typedef struct
{
  BYTE                OpCode;
  BYTE                Reserved;
} TFOEHEADER;

typedef struct
{
  TMBXHEADER          MbxHeader;
  TFOEHEADER          FoeHeader;
  DWORD               Password;
  char                FileName[MAX_FILE_NAME_SIZE};
} TFOEREADREQ;
```

Figure 39 – Read Request structure

The FoE Read Request coding is specified in Table 88.

Table 88 – Read Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N > 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x04: FoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
FoE Header	OpCode	BYTE	0x01: Read Request
	Reserved	BYTE	Shall be zero
Read Header	Password	DWORD	0: password unused 1-0xFFFFFFFF: password
	File Name	char[n-6]	Name of the file to be read

5.8.2 Write Request

The attribute types of Write Request are described in Figure 40.

```
typedef struct
{
  BYTE                OpCode;
  BYTE                Reserved;
} TFOEHEADER;

typedef struct
{
  TMBXHEADER          MbxHeader;
  TFOEHEADER          FoeHeader;
  DWORD               Password;
  char                FileName[MAX_FILE_NAME_SIZE};
} TFOEWRITEREQ;
```

Figure 40 – Write Request structure

The FoE Write Request coding is specified in Table 89.

Table 89 – Write Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N > 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x04: FoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
FoE Header	OpCode	BYTE	0x02: Write Request
	Reserved	BYTE	Shall be zero
Write Header	Password	DWORD	0: password unused 1-0xFFFFFFFF: password
	File Name	char[n-6]	Name of the file to be written

5.8.3 Data Request

The attribute types of Data Request are described in Figure 41.

```
typedef struct
{
  BYTE               OpCode;
  BYTE               Reserved;
} TFOEHEADER;

typedef struct
{
  TMBXHEADER         MbxHeader;
  TFOEHEADER         FoeHeader;
  DWORD              PacketNo;
  BYTE               Data[MAX_DATA_SIZE};
} TFOEDATAREQ;
```

Figure 41 – Data Request structure

The FoE Data Request coding is specified in Table 90.

Table 90 – Data Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N >= 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x04: FoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
FoE Header	OpCode	BYTE	0x03: Data Request
	Reserved	BYTE	Shall be zero
Data Header	Packet Number	DWORD	1-0xFFFFFFFF
	Data	BYTE[n-6]	File data

5.8.4 Ack Request

The attribute types of Ack Request are described in Figure 42.

```
typedef struct
{
  BYTE                OpCode;
  BYTE                Reserved;
} TFOEHEADER;

typedef struct
{
  TMBXHEADER          MbxHeader;
  TFOEHEADER          FoeHeader;
  DWORD               PacketNo;
} TFOEACKREQ;
```

Figure 42 – Ack Request structure

The FoE Ack Request coding is specified in Table 91.

Table 91 – Ack Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x04: FoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
FoE Header	OpCode	BYTE	0x04: Ack Request
	Reserved	BYTE	Shall be zero
Ack Header	Packet Number	DWORD	0: acknowledge of Write Request 1-0xFFFFFFFF: acknowledge of Data Request

5.8.5 Error Request

The attribute types of Error Request are described in Figure 43.

```
typedef struct
{
  BYTE                OpCode;
  BYTE                Reserved;
} TFOEHEADER;

typedef struct
{
  TMBXHEADER          MbxHeader;
  TFOEHEADER          FoeHeader;
  DWORD               ErrorCode;
  char                ErrorText[MAX_ERROR_TEXT_SIZE};
} TFOEERRORREQ;
```

Figure 43 – Error Request structure

The FoE Error Request coding is specified in Table 92.

Table 92 – Error Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N >= 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x04: FoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
FoE Header	OpCode	BYTE	0x05: Error Request
	Reserved	BYTE	Shall be zero
Error Header	Error Code	DWORD	1-0xFFFFFFFF
	Error Text	char[n-6]	Optional error description

The error codes of FoE are specified in Table 93.

Table 93 – Error codes of FoE

error code	meaning
0x8000	Not defined
0x8001	Not found
0x8002	Access denied
0x8003	Disk full
0x8004	Illegal
0x8005	Packet number wrong
0x8006	Already exists
0x8007	No user
0x8008	Bootstrap only
0x8009	Not Bootstrap
0x800A	No rights
0x800B	Program Error

5.8.6 Busy Request

The attribute types of Busy Request are described in Figure 44.

```
typedef struct
{
  BYTE              OpCode;
  BYTE              Reserved;
} TFOEHEADER;

typedef struct
{
  TMBXHEADER        MbxHeader;
  TFOEHEADER        FoeHeader;
  WORD              Done;
  WORD              Entire;
  char              BusyText[MAX_BUSY_TEXT_SIZE};
} TFOEBUSYREQ;
```

Figure 44 – Busy Request structure

The FoE Busy Request coding is specified in Table 94.

Table 94 – Busy Request

Frame part	Data Field	Data Type	Value/Description
Mailbox Header	Length	WORD	N >= 0x06: Length of the Mailbox Service Data
	Address	WORD	Station Address of the source, if a master is client, Station Address of the destination, if a slave is client
	Channel	Unsigned6	0x00 (Reserved for future)
	Priority	Unsigned2	0x00: lowest priority … 0x03: highest priority
	Type	Unsigned4	0x04: FoE
	Cnt	Unsigned3	Counter of the mailbox services (0 reserved, 1 is start value, next value after 7 is 1)
	Reserved	Unsigned1	0x00
FoE Header	OpCode	BYTE	0x06: Busy Request
	Reserved	BYTE	Shall be zero
Busy Header	Done	WORD	If entire is “0”, Done indicates the progress in percent. 0-100 (done in %) If entire is unequal to “0” the value indicates the size of the already transferred data in the same unit as the element “Entire”.
	Entire	WORD	If element is “0” the Done element indicates progress of the file transfer from 0% to 100%. If unequal to “0” Entire indicates the file size of the actual transfer in a specified unit. In this case Done indicates the size of date which already has been transferred in the same unit. With help of this two values a percentage quotation can be calculated: 100*Done/Entire
	Busy Text	char[n-6]	Optional busy description

6 FAL protocol state machines

6.1 Overall structure

6.1.1 Overview

The FAL protocol state machine structure is as defined in Figure 45.

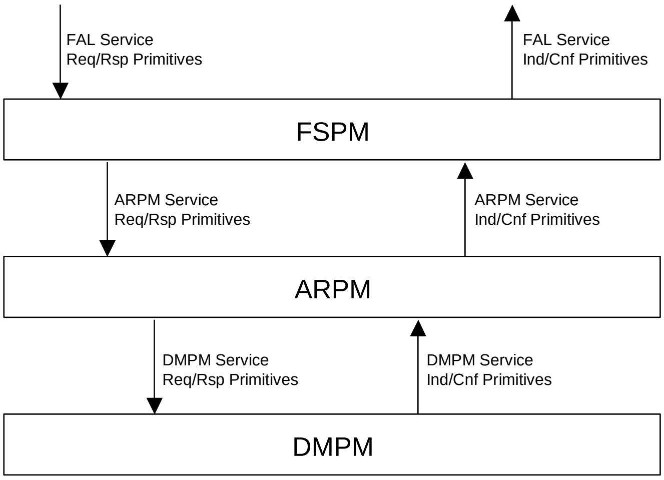

Figure 45 – Relationship among Protocol Machines

The behavior of the FAL is specified by three integrated protocol machines. The FSPM is the service interface between the FAL services that are part of the FAL Class specification and the particular AREP.

The EtherCAT FAL provides a set of protocol machines for slave. The masters can anticipate the behavior of the slaves.

The FSPM is responsible for the following activities:

- To accept service primitives from the FAL service user and convert them into FAL internal primitives.
- To select the ARPM state machine based on the implicit addressing mechanism and send FAL internal primitives with the service parameters to the ARPM.
- To accept FAL internal primitives from the ARPM and convert them into service primitives for the FAL service user.
- To deliver the FAL service primitives to the FAL user.

The ARPM specifies the conveyance type for the application relation.

The DMPM specifies the mapping to the Data Link Layer. The DMPM defines therefore two protocol machines, the LMPM and the MAC protocol machines.

6.1.2 Fieldbus Service Protocol Machines (FSPM)

The FSPM State Machines co-ordinate the underlying state machines used for processing of the various services and application relations.

The FSPM basically is a mapping protocol machine. The main task is to pass the service to the protocol machine responsible for that service and to forward confirmations and responses to the user. In addition a basic redundancy control scheme is included in this machine that allows to collaborate two AR into a single entity with higher availability.

6.1.3 Application Relationship Protocol Machines (ARPM)

The ARPMs are responsible for the individual service procedures execution. The overall structure is shown in Figure 46. Process Data interaction is directly handled by DL and controlled by ESM. There are various ways for the application to run mailbox protocols.

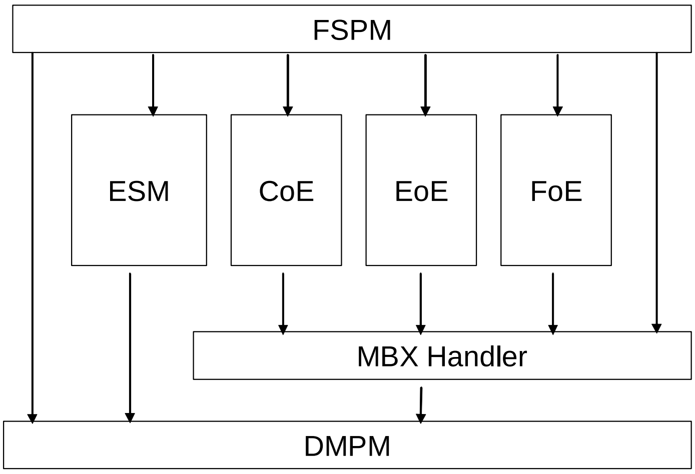

Figure 46 – AR Protocol machines

6.1.4 DLL Mapping Protocol Machines (DMPM)

The DL Mapping Protocol Machines (DMPM) connects the other State machines and Layer 2. DMPM provides the coordination of all state machines concerning the configuration and error handling of the Data Link Layer Usage. The functions are mapped by the DMPM to the DLL services of Layer 2. The DMPM generates the necessary Layer 2 parameters of the service, receives the confirmations and indications from Layer 2 and passes them to the appropriate DMPM-User.

6.2 AP-Context state machine

There is no AP-Context State Machine defined for this Protocol.

6.3 FAL service protocol machine (FSPM)

The services specified in ETG.1000.5 are directly mapped to services of the ARPMs.

6.4 Application Relationship Protocol Machines (ARPMs)

6.4.1 AL state machine

6.4.1.1 Description

ESM is responsible for the coordination of master and slave at start up and during operation. State changes are mainly caused by interactions between master and slave. They are primarily related to writes to AL Control word.

After Initialization of DL and AL the machine enters the INIT State. The ‘Init’ state defines the root of the communication relationship between the master and the slave in application layer. No direct communication between the master and the slave on application layer is possible. The master uses the ‘Init’ state to initialize a set of configuration register. If the slave supports a mailbox, the corresponding sync manager configurations are also done in the ‘Init’ state.

The ‘Pre-Operational’ state can be entered if the settings of the mailbox have been done if the slave supports the optional mailbox. Both the master and the slave can use the mailbox and the appropriate protocols to exchange application specific initializations and parameters. No process data communication is possible in this state.

The ‘Safe-Operational’ state can be entered if the settings of the input buffer have been done if the slave supports the inputs and the master requests inputs. The application of the slave shall deliver actual input data without processing the output data. The real outputs of the slave shall be set to their “safe state”.

The ‘Operational’ state can be entered if the settings of the output buffer have been done and actual outputs have been delivered to the slave (provides outputs of the slave will be used).

The application of the slave shall deliver actual input data and the application of the master shall provide output data.

In the optional ‘Bootstrap’ state the application of the slave shall be able to accept persistent settings downloaded with the FoE protocol.

The ESM defines four states, which shall be supported:

- Init,
- Pre-Operational,
- Safe-Operational, and
- Operational.

All state changes are possible except for the ‘Init’ state, where only the transition to the ‘Pre-Operational’ state is possible and for the ‘Pre-Operational’ state, where no direct state change to ‘Operational’ exists.

State changes are normally requested by the master. The master requests a write to the AL Control register which results in a Register Event ‘AL Control’ indication in the slave. The slave shall respond to the change in AL Control through a local AL Status write service after a successful or a failed state change. If the requested state change failed, the slave shall respond with the error flag set.

EtherCAT devices can either be so called simple devices or complex devices. Their behaviour regarding AL control request and AL control response is different. While complex slaves reset the AL Error Flag if possible on receive of an AL Acknowledge flag (device emulation inactive) simple slaves will copy the Acknowledge flag to the AL Error flag (device emulation active).

Despite of the different behaviour a master should have the possibility to initialize the network by a broadcast command. Hence, it shall be allowed to reset all devices by sending a broadcast INIT request with Ack Flag set to false (AL Control register = 0x0001). Complex devices shall then reset the Error Flag.

In case of an error first the AL Status Code shall be set and then the Error Flag has to be set. After clearing the Error Flag the AL Status Code should be cleared, too. The master shall ignore the AL Status Code if the Error Flag is clear.

In case of a state transition from AL state Op to SafeOp with Error the output SyncManager shall be disabled. The input SyncManager shall only be disabled in case of an error which results in invalid inputs (input error or synchronization error).

The Output-SyncManager shall be re-enabled if the error was acknowledged and there is no output error remaining.

The Input-SyncManager shall be re-enabled if the error was acknowledged and there is no input error remaining.

The Bootstrap state is optional and there is only a transition from or to the Init state. The only purpose of this state is to download the device’s firmware. In Bootstrap state the mailbox is active but restricted to the FoE protocol.

ESM is specified in Figure 47.

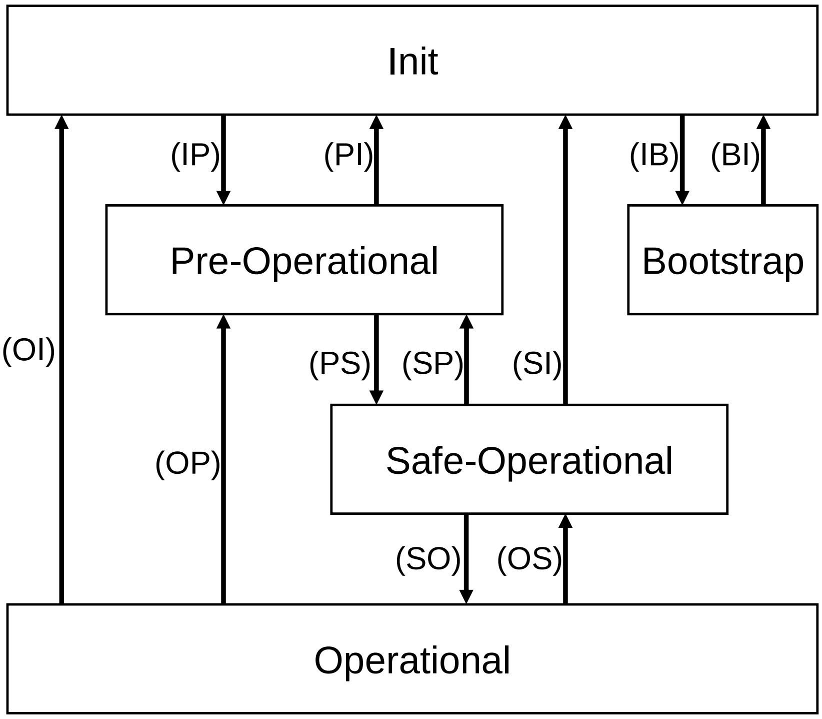

Figure 47 – ESM Diagramm

The local management services are related to the transitions in the ESM, as specified in Table 95. If there is more than one service related to the transition, the slave’s application will process all of the related services.

Table 95 – State transitions and local management services

state transition	local management service
IP	Start Mailbox Communication
PI	Stop Mailbox Communication
PS	Start Input Update
SP	Stop Input Update
SO	Start Output Update
OS	Stop Output Update
OP	Stop Output Update, Stop Input Update
SI	Stop Input Update, Stop Mailbox Communication
OI	Stop Output Update, Stop Input Update, Stop Mailbox Communication
IB	Start Bootstrap Mode
BI	Restart Device

6.4.1.2 ESM States

6.4.1.2.1 Init

The ‘Init’ state defines the root of the communication relationship between the master and the slave in application layer. No direct communication between the master and the slave on application layer is possible. The master uses the ‘Init’ state to initialize a set of configuration register of the ESC. If the slave supports mailbox services, the corresponding sync manager configurations are also done in the ‘Init’ state.

6.4.1.2.2 Pre-Operational

In the ‘Pre-Operational’ state the mailbox is active if the slave supports the optional mailbox. Both the master and the slave can use the mailbox and the appropriate protocols to exchange application specific initializations and parameters. No process data communication is possible in this state.

6.4.1.2.3 Safe-Operational

In the ‘Safe-Operational’ state the application of the slave shall deliver actual input data without manipulating the output data. The outputs shall be set to their “safe state”.

6.4.1.2.4 Operational

In the ‘Operational’ state the application of the slave shall deliver actual input data and application of the master shall deliver actual output data.

6.4.1.2.5 Bootstrap

In the optional ‘Bootstrap’ state the application of the slave shall be able to accept a new firmware downloaded with the FoE protocol.

6.4.1.3 Primitive definitions

6.4.1.3.1 Primitives exchanged between DL and ESM

Table 96 shows the service primitives including their associated parameters issued by the ESM and received by the DL.

Table 96 – Primitives issued by ESM to DL

Primitive name	Associated parameters	Functions
AL State Change.req	AL Status Application Specific AL Status Code	Refer to Service Definition ETG.1000.5
AL Control.rsp(+)	AL State AL Status Code	Refer to Service Definition ETG.1000.5
AL Control.rsp(-)	AL State AL Status Code	Refer to Service Definition ETG.1000.5

Table 97 shows the service primitives including their associated parameters issued by the DL received by the ESM.

Table 97 – Primitives issued by DL to ESM

Primitive name	Associated parameters	Functions
AL Control.ind	AL Control State Ack Flag ID Request	Refer to Service Definition ETG.1000.5

6.4.1.3.2 Primitives exchanged between Application and ESM

Table 98 shows the service primitives including their associated parameters issued by the AL and received by the ESM.

Table 98 – Primitives issued by Application to ESM

Primitive name	Associated parameters	Functions
Stop Input		Application stops update of process data
SM Change		Sync Manager Configuration change (can be enabled/disabled locally or issued by communication)

Table 99 shows the service primitives including their associated parameters issued by the ESM received by the AL.

Table 99 – Primitives issued by ESM to Application

Primitive name	Associated parameters	Functions
START_MBX_HANDLER		Start Mailbox Communication INIT->PREOP: Start Mailbox Communication by enabling SyncManager
STOP_MBX_HANDLER		Stop Mailbox Communication PREOP->INIT: Stop Mailbox Communication by disabling SyncManager
START_INPUT_HANDLER		Start Input Update PREOP->SAFEOP: Start Input Update to the ESC by enabling SyncManager Start Output Update from the ESC by enabling SyncManager
STOP_INPUT_HANDLER		Stop Input Update SAFEOP->PREOP: Stop Input Update to the ESC by disabling SyncManager Stop Output Update from the ESC by disabling SyncManager Stop watchdog
START_LOCAL_OUTPUT_HANDLER		Start local Output Update SAFEOP->OP: Set Outputs to received process values
STOP_LOCAL_OUTPUT_HANDLER		Stop local Output Update OP->SAFEOP: Set Outputs to safe values

ID_INFO		If ID Request AND IdSupported then
		AL_STATUS_CODE= ID value
		ID_FLAG = 1
		else
		ID_FLAG = 0
		end_if

6.4.1.3.3 ESM Variables

The Table 100 defines the variables used in the ESM.

Table 100 – ESM Variables

Vaiable name	Description
bootSupported	Bootstrap mode is supported by slave

Vaiable name	Description
dcNotAccepted	The device does not support DCs and does not accept DC settings, either. If DC settings are made the device will go into Error state. This means Sync Control register 0x0981 shall be 0.
dcRequired	Device requires DC settings (Bits 0x0981:00 to 0x0981:02 shall be set accordingly). Other modes such as “Freerun” and “Synchronous with SM event” are not supported
dcRunning	DCs are running and used for synchronization between master ans slave application.
dcSupported	Device supports at least one OpMode which uses Distributed Clocks
IdSupported	Device supports Explicit Device Identification using Register 0x0134
localErrorCode	Original reason of a previous local error which resulted in disabeling SyncManager channels
localErrorFlag	Flag of local application indicating a local error causing that SyncManagers were disabled. Used in case of an error resulting in a state change from Op to SafeOp
pllRunning	Local application is synchronized with DC event and master application.
safeOp2OpTimeoutTimer	Timeout for state change from SafeOp to Op
dcEventReceived	Slave has received the Sync0/Sync1 event at least once
waitForPllRunning	Wait until local PLL has locked with master cycle (wait until pllRunning = TRUE). This time shall be smaller than the SafeOp2OpTimeout
wdEnabled	The watchdog behavior is disabled if at least the watchdog time (R400, R420) is equal zero, additionally the watchdog behavior can be disabled if the watchdog bit of the SyncManager is disabled (SyncManager register +0x04.06=0) (when using the watchdog functionality of the ESC, this watchdog is only enabled if the watchdog bit in the Sync Manager is set) TRUE: watchdog behaviour is enabled FALSE: watchdog behaviour is disabled If slave has only inputs wdEnabled may always be FALSE
readyForOP	Internal variable which is set when the device is ready to be switched to OP in Non-DC-Mode, this should happen when the outputs have been received within the watchdog time if enabled or during SafeOp state when the watchdog is disabled. However there can be another behavior due to legacy reasons.
smEventReceived	Internal variable which indicates that the SM-event was received

6.4.1.3.4 ESM Macros

The Table 101 defines the macros used in the ESM.

Table 101 – ESM macros

Macro name	Description
SM_SETTINGS_0_1_MATCH	local function that checks requested Mailbox-SyncManager settings against local settings
SM_SETTINGS_2_TO_n_MATCH	local function that checks requested Process data SyncManager settings against local settings For slaves without Mailbox this can be SM 0 to n.

Macro name	Description
DC_ACTIVATED	Local application is synchronized. NOTE: AssignActivate shall be used with a logical AND operation with 0x0701 and be either 0x0300 or 0x0700 (all other bits can be set as required)
DISABLE_SM_CHANNEL	SyncManager 2 shall be disabled. Only if there is no Output SyncManager or an input error occurs the Input SyncManager shall be disabled while SyncManager 2 remains enabled.
ENABLE_SM_CHANNEL	SyncManagers shall be enabled
RESTART_WD	If watchdog is enabled: Watchdog of ESC is started. Additionally, a local watchdog timer on the host controller may be started.

6.4.1.3.5 ESM Functions

The Table 102 defines the functions used in the ESM.

Table 102 – ESM functions

Function name	Description
Output Event	Primitve issued from DL to ESM: Event that indicates arrival of new output data
WD Expired	(Local) WD is expired
SM_Chg	Primitve issued from DL to ESM. Event service of DL to indicate a change in the SM settings
AL_State Change.req (AL Status, AL Status Code)	Primitve issued from Application to ESM: Slave application indicates a state change
PLL Running Event	Local event that indicates that the master is now synchronized to the local application (outputs are sent by the master and received by the slave before the DC Event)
SafeOp2OpTimeoutTimer Expired	Event indicating that the local state change timeout has expired
Start SafeOp2OpTimeoutTimer	Local timeout for SafeOp to Op timeout is started
AckSM_Chg	SyncManager Change Event shall be acknowledged by the slave
Stop Inp	Local Function of AL to disable Input Update
SM_SETTINGS_OBJECT_SYNC_TYPE_MATCH	local function that checks requested Synchronization settings with local properties

6.4.1.3.6 Parameters of primitives

The parameters used with the primitives exchanged between the DL, ESM and the Application are described in ETG.1000.5.

6.4.1.4 ESM State Table

Table 103 contains the complete description of the ESM state machine.

Table 103 – ESM state table

#	Current State	Event /Condition =>Action	Next State
1.1	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_ERROR_FLAG = 1 and .ACK_FLAG = 0 and AL_CONTROL_STATE = STATE_INIT => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_INIT ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
1.2	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_ERROR_FLAG = 1 and .ACK_FLAG = 0 and AL_CONTROL_STATE <> STATE_INIT => AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
2	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_INIT => AL_ERROR_FLAG = 0 ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
3	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_PREOP and SM_SETTINGS_0_AND_1_MATCH => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_PREOP START_MBX_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP

4	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		AL_CONTROL_STATE = STATE_PREOP and not
		SM_SETTINGS_0_AND_1_MATCH
		=>
		ID_FLAG = 0
		AL_STATE = STATE_INIT
		AL_STATUS_CODE = 0x16
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			INIT

5	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_BOOT and BOOT_SUPPORTED and SM_SETTINGS_0_AND_1_MATCH => AL_ERROR_FLAG = 0 AL_STATE = STATE_BOOT AL_STATUS_CODE = 0 START_MBX_HANDLER AL Control.rsp(+) (AL State, AL Status Code, Error Flag)	BOOT

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State

6	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		AL_CONTROL_STATE = STATE_BOOT and BOOT_SUPPORTED and not SM_SETTINGS_0_AND_1_MATCH
		=>
		ID_FLAG = 0
		AL_STATE = STATE_INIT
		AL_STATUS_CODE = 0x15
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			INIT

7	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		AL_CONTROL_STATE = STATE_BOOT and not BOOT_SUPPORTED
		=>
		ID_FLAG = 0
		AL_STATE = STATE_INIT
		AL_STATUS_CODE = 0x13
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			INIT

8	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		(AL_CONTROL_STATE = STATE_SAFEOP OR
		AL_CONTROL_STATE = STATE_OP)
		=>
		ID_FLAG = 0
		AL_STATE = STATE_INIT
		AL_STATUS_CODE = 0x11
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			INIT

9	INIT	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and (AL_CONTROL_STATE = unknownState) => ID_FLAG = 0 AL_STATE = STATE_INIT AL_STATUS_CODE = 0x12 AL_ERROR_FLAG = 1 AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
10	INIT	**SM_Chg** => ignore	INIT
11.1	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_ERROR_FLAG = 1 and .ACK_FLAG = 0 and AL_CONTROL_STATE = STATE_INIT => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_INIT STOP_MBX_HANDLER ID_INFO AL Control.rsp(~~-~~ +) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
11.2	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_ERROR_FLAG = 1 and .ACK_FLAG = 0 and AL_CONTROL_STATE <> STATE_INIT => AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
12	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_INIT => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_INIT STOP_MBX_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
13	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_PREOP => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP
14.1	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_SAFEOP and SM_SETTINGS_2_TO_n_MATCH and ((not DC_ACTIVATED and not dcRequired) or (DC_ACTIVATED and not dcNotAccepted and not dcSupported)) => dcRunning = FALSE AL_ERROR_FLAG = 0 AL_STATUS_CODE =0 AL_STATE = STATE_SAFEOP START_INPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
14.2	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_SAFEOP and SM_SETTINGS_2_TO_n_MATCH and DC_ACTIVATED and not dcNotAccepted and dcSupported => dcRunning = TRUE AL_ERROR_FLAG = 0 AL_STATUS_CODE =0 AL_STATE = STATE_SAFEOP START_INPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
14.3	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_SAFEOP and SM_SETTINGS_2_TO_n_MATCH and ((DC_ACTIVATED and dcNotAccepted) or (not DC_ACTIVATED and dcRequired)) => ID_FLAG = 0 AL_STATUS_CODE = 0x30 AL_ERROR_FLAG = 1 AL_STATE = STATE_PREOP AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State

17	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		AL_CONTROL_STATE = STATE_SAFEOP and not
		SM_SETTINGS_2_TO_n_MATCH
		=>
		ID_FLAG = 0
		AL_STATE = STATE_PREOP
		AL_STATUS_CODE = 0x17
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			PREOP

18	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		(AL_CONTROL_STATE = STATE_BOOT or
		AL_CONTROL_STATE = STATE_OP)
		=>
		ID_FLAG = 0
		AL_STATE = STATE_PREOP
		AL_STATUS_CODE = 0x11
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			PREOP

19	PREOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and (AL_CONTROL_STATE = unknownState) => ID_FLAG = 0 AL_STATE = STATE_PREOP AL_STATUS_CODE = 0x12 AL_ERROR_FLAG = 1 AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP
20.1	PREOP	**SM_Chg** / AL_ERROR_FLAG = 0 and SM_SETTINGS_0_AND_1_MATCH => AckSM_Chg	PREOP
20.2	PREOP	**SM_Chg** / AL_ERROR_FLAG = 1 => ignore	PREOP
21	PREOP	**SM_Chg** /AL_ERROR_FLAG = 0 and not SM_SETTINGS_0_AND_1_MATCH => ID_FLAG = 0 AL_STATUS_CODE = 0x16 AL_ERROR_FLAG = 1 AL_STATE = STATE_INIT STOP_MBX_HANDLER AckSM_Chg AL Status Changed.req (AL state, AL staus code, Error Flag, ID Flag)	INIT
22.1	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 1 and ACK_FLAG = 0) and AL_CONTROL = STATE_INIT => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_INIT STOP_MBX_HANDLER STOP_INPUT_HANDLER STOP_LOCAL_OUTPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	INIT

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
22.2	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 1 and ACK_FLAG = 0) and AL_CONTROL_STATE <> STATE_INIT => AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
23	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_INIT => AL_ERROR_FLAG = 0 AL_STATE = STATE_INIT STOP_MBX_HANDLER STOP_INPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	INIT
24	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_PREOP => AL_ERROR_FLAG = 0 AL_STATE = STATE_PREOP STOP_INPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP
25.1	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 1 and (not localErrorFlag and .ACK_FLAG = 1)) and AL_CONTROL_STATE = STATE_SAFEOP => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 ENABLE_SM_CHANNEL ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
25.2	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /((AL_ERROR_FLAG = 0 or (localErrorFlag and .ACK_FLAG = 1)) and AL_CONTROL_STATE = STATE_SAFEOP => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
26.2	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_OP and readyForOP and (not dcRunning or pllRunning) and not localErrorFlag => AL_ERROR_FLAG = 0 AL_STATUS_CODE =0 AL_STATE = STATE_OP ENABLE_SM_CHANNEL START_LOCAL_OUTPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	OP

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
26.4	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_OP and dcRunning and not pllRunning and not localErrorFlag => ENABLE_SM_CHANNEL waitForPllRunning = TRUE Start SafeOp2OpTimeoutTimer ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
26.5	SAFEOP	**PLL Running Event** /waitForPllRunning => AL_STATE = STATE_OP AL_ERROR_FLAG = 0 AL_STATUS_CODE =0 START_LOCAL_OUTPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	OP
26.6	SAFEOP	**PLL Running Event** /not waitForPllRunning => ignore	SAFEOP

26.7	SAFEOP	**SafeOp2OpTimeoutTimer Expired**
		/waitForPllRunning and not dcEventReceived
		=>
		ID_FLAG = 0
		AL_STATE = STATE_SAFEOP
		AL_STATUS_CODE = 0x2D
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			SAFEOP

26.8	SAFEOP	**SafeOp2OpTimeoutTimer Expired**
		/waitForPllRunning and dcEventReceived
		=>
		ID_FLAG = 0
		AL_STATE = STATE_SAFEOP
		AL_STATUS_CODE = 0x32
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			SAFEOP

26.9	SAFEOP	**SafeOp2OpTimeoutTimer Expired** /not waitForPllRunning => ignore	SAFEOP
27.1	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and AL_CONTROL_STATE = STATE_OP and not readyForOP and (not dcRunning or pllRunning) and not localErrorFlag => ID_FLAG = 0 AL_STATE = STATE_SAFEOP AL_STATUS_CODE = 0x1B AL_ERROR_FLAG = 1 AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State

27.2	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		AL_CONTROL_STATE = STATE_OP and
		localErrorFlag
		=>
		ID_FLAG = 0
		AL_STATE = STATE_SAFEOP
		AL_STATUS_CODE = localErrorCode
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			SAFEOP

29	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)**
		/(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and
		AL_CONTROL_STATE = STATE_BOOT
		=>
		ID_FLAG = 0
		AL_STATE = STATE_SAFEOP
		AL_STATUS_CODE = 0x11
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)
			SAFEOP

30	SAFEOP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and (AL_CONTROL_STATE = unknownState) => ID_FLAG = 0 AL_STATE = STATE_SAFEOP AL_STATUS_CODE = 0x12 AL_ERROR_FLAG = 1 AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
31.1	SAFEOP	**SM_Chg** / AL_ERROR_FLAG = 0 and SM_SETTINGS_0_AND_1_MATCH and SM_SETTINGS_2_TO_n_MATCH => AckSM_Chg	SAFEOP
31.2	SAFEOP	**SM_Chg** / AL_ERROR_FLAG = 1 => ignore	SAFEOP
32	SAFEOP	**SM_Chg** / AL_ERROR_FLAG = 0 and SM_SETTINGS_0_AND_1_MATCH and not SM_SETTINGS_2_TO_n_MATCH => ID_FLAG = 0 AL_STATUS_CODE = 0x17 AL_ERROR_FLAG = 1 AL_STATE = STATE_PREOP STOP_INPUT_HANDLER AckSM_Chg AL Status Changed.req (AL state, AL staus code)	PREOP
33	SAFEOP	**SM_Chg** / AL_ERROR_FLAG = 0 and not SM_SETTINGS_0_AND_1_MATCH => ID_FLAG = 0 AL_STATUS_CODE = 0x16 AL_ERROR_FLAG = 1 AL_STATE = STATE_INIT STOP_INPUT_HANDLER STOP_MBX_HANDLER AckSM_Chg AL Status Changed.req (AL state, AL staus code, Error Flag, ID Flag)	INIT

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State

34.1	SAFEOP	**AL_State Change.req (AL Status, AL Status Code)**
		/AL_STATE = STATE_SAFEOP and AL_ERROR_FLAG = 1
		=>
		ID_FLAG = 0
		DISABLE_SM_CHANNEL
		AL_STATE = STATE_SAFEOP
		AL_ERROR_FLAG = TRUE
		localErrorFlag = TRUE
		localErrorCode = AL_STATUS_CODE
		AL Status Changed.req (AL state, AL Status Code, Error Flag, ID Flag)
			SAFEOP

34.2	SAFEOP	**AL_State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_SAFEOP and AL_ERROR_FLAG = 0 and localErrorFlag => ENABLE_SM_CHANNEL localErrorFlag = FALSE localErrorCode = 0	SAFEOP
34.3	SAFEOP	**AL_State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_PREOP and AL_ERROR_FLAG = 1 => ID_FLAG = 0 STOP_INPUT_HANDLER AL_STATE = STATE_PREOP AL_ERROR_FLAG = TRUE AL Status Changed.req (AL state, AL Status Code, Error Flag, ID Flag)	PREOP
34.4	SAFEOP	**AL_State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_INIT and AL_ERROR_FLAG = 1 => ID_FLAG = 0 STOP_MBX_HANDLER STOP_INPUT_HANDLER AL_STATE = STATE_INIT AL_ERROR_FLAG = TRUE AL Status Changed.req (AL state, AL Status Code, Error Flag, ID Flag)	INIT
34.5	SAFEOP	**AL_State Change.req (AL Status, AL Status Code)** /((AL_STATE = STATE_INIT or STATE_PREOP) and AL_ERROR_FLAG = 0) or AL_STATE = STATE_OP or STATE_BOOT or unknown => ignore	SAFEOP
35.1	SAFEOP	**Output Event** => RESTART_WD	SAFEOP
37	OP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_CONTROL_STATE = STATE_INIT => AL_STATUS_CODE =0 AL_ERROR_FLAG = 0 AL_STATE = STATE_INIT STOP_MBX_HANDLER STOP_INPUT_HANDLER STOP_LOCAL_OUTPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	INIT

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
38	OP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_CONTROL_STATE = STATE_PREOP => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_PREOP STOP_INPUT_HANDLER STOP_LOCAL_OUTPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	PREOP
39	OP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_CONTROL_STATE = STATE_SAFEOP => AL_ERROR_FLAG = 0 AL_STATUS_CODE = 0 AL_STATE = STATE_SAFEOP STOP_LOCAL_OUTPUT_HANDLER ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
40	OP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_CONTROL_STATE = STATE_OP => ID_INFO AL Control.rsp(+) (AL State, AL Status Code, Error Flag, ID Flag)	OP
42	OP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_CONTROL_STATE = STATE_BOOT => ID_FLAG = 0 AL_STATE = STATE_SAFEOP AL_STATUS_CODE = 0x11 AL_ERROR_FLAG = 1 DISABLE_SM_CHANNEL STOP_LOCAL_OUTPUT_HANDLER AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
43	OP	**AL_Control.ind (AL Control State, Ack Flag, ID Request)** /AL_CONTROL_STATE = unknownState => ID_FLAG = 0 AL_STATE = STATE_SAFEOP AL_STATUS_CODE = 0x12 AL_ERROR_FLAG = 1 DISABLE_SM_CHANNEL STOP_LOCAL_OUTPUT_HANDLER AL Control.rsp(-) (AL State, AL Status Code, Error Flag, ID Flag)	SAFEOP
44	OP	**SM_Chg** / AL_ERROR_FLAG = 0 and SM_SETTINGS_0_AND_1_MATCH and SM_SETTINGS_2_TO_n_MATCH => AckSM_Chg	OP

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
45	OP	**SM_Chg** / AL_ERROR_FLAG = 0 and SM_SETTINGS_0_AND_1_MATCH and not SM_SETTINGS_2_TO_n_MATCH => ID_FLAG = 0 AL_STATUS_CODE = 0x17 AL_ERROR_FLAG = 1 AL_STATE = STATE_PREOP STOP_LOCAL_OUTPUT_HANDLER STOP_INPUT_HANDLER AckSM_Chg AL Status Changed.req (AL state, AL staus code, Error Flag, ID Flag)	PREOP
46	OP	**SM_Chg** / AL_ERROR_FLAG = 0 and not SM_SETTINGS_0_AND_1_MATCH => ID_FLAG = 0 AL_STATUS_CODE = 0x16 AL_ERROR_FLAG = 1 AL_STATE = STATE_INIT STOP_LOCAL_OUTPUT_HANDLER STOP_INPUT_HANDLER STOP_MBX_HANDLER AckSM_Chg AL Status Changed.req (AL state, AL staus code, Error Flag, ID Flag)	INIT
47.1	OP	**AL State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_SAFEOP and AL_ERROR_FLAG = 1 => STOP_LOCAL_OUTPUT_HANDLER DISABLE_SM_CHANNEL AL_STATE = STATE_SAFEOP localErrorFlag = TRUE localErrorCode = AL_STATUS_CODE AL Status Changed.req (AL state, AL Status Code, Error Flag, ID Flag)	SAFEOP
47.2	OP	**AL State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_PREOP and AL_ERROR_FLAG = 1 => ID_FLAG = 0 STOP_INPUT_HANDLER STOP_LOCAL_OUTPUT_HANDLER AL_STATE = STATE_PREOP localErrorFlag = TRUE localErrorCode = AL_STATUS_CODE AL Status Changed.req (AL state, AL Status Code, Error Flag, ID Flag)	PREOP
47.3	OP	**AL State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_INIT and AL_ERROR_FLAG = 1 => ID_FLAG = 0 STOP_MBX_HANDLER STOP_INPUT_HANDLER STOP_LOCAL_OUTPUT_HANDLER AL_STATE = STATE_INIT localErrorFlag = TRUE localErrorCode = AL_STATUS_CODE AL Status Changed.req (AL state, AL status code, Error Flag, ID Flag)	INIT
47.4	OP	**AL State Change.req (AL Status, AL Status Code)** /((AL_STATE = STATE_SAFEOP or STATE_PREOP or STATE_INIT) and AL_ERROR_FLAG = 0) or AL_CONTROL_STATE = STATE_BOOT or unknown => ignore	OP

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
48	OP	**Output event** => RESTART_WD Update Outputs	OP
49	OP	**WD expired** => ID_FLAG = 0 AL_STATUS_CODE = 0x1B AL_ERROR_FLAG = 1 AL_STATE = STATE_SAFEOP STOP_LOCAL_OUTPUT_HANDLER DISABLE_SM_CHANNEL AL Status Changed.req (AL state, AL staus code, Error Flag, ID Flag)	SAFEOP
51	BOOT	**AL_Control.ind (AL Control State, Ack Flag)** /AL_CONTROL_STATE = STATE_INIT => AL_ERROR_FLAG = 0 AL_STATE = STATE_INIT STOP_MBX_HANDLER AL Control.rsp(+) (AL State, AL Status Code)	INIT
52	BOOT	**AL_Control.ind (AL Control State, Ack Flag)** / AL_CONTROL_STATE = STATE_BOOT => AL_ERROR_FLAG = 0 AL Control.rsp(+) (AL State, AL Status Code)	BOOT

53.1	BOOT	**AL_Control.ind (AL Control State, Ack Flag)**
		/(AL_CONTROL_STATE = STATE_PREOP or
		AL_CONTROL_STATE = STATE_SAFEOP or
		AL_CONTROL_STATE = STATE_OP)
		=>
		AL_STATUS_CODE = 0x11
		AL_ERROR_FLAG = 1
		AL Control.rsp(-) (AL State, AL Status Code)
			INIT

53.2	BOOT	**AL_Control.ind (AL Control State, Ack Flag)** /(AL_CONTROL_STATE = STATE_PREOP or AL_CONTROL_STATE = STATE_SAFEOP or AL_CONTROL_STATE = STATE_OP) => ignore	BOOT
54.1	BOOT	**AL_Control.ind (AL Control State, Ack Flag)** /(AL_CONTROL_STATE = unknownState) => AL_STATUS_CODE = 0x12 AL_ERROR_FLAG = 1 AL Control.rsp(-) (AL State, AL Status Code)	INIT
54.2	BOOT	**AL_Control.ind (AL Control State, Ack Flag)** /(AL_ERROR_FLAG = 0 or .ACK_FLAG = 1) and (AL_CONTROL_STATE = unknownState) => ignore	BOOT
55	BOOT	**SM_Chg** /SM_SETTINGS_0_AND_1_MATCH => ignore	BOOT

Table 103 *(continued)*

#	Current State	Event /Condition =>Action	Next State
56.1	BOOT	**SM_Chg** /not SM_SETTINGS_0_AND_1_MATCH => AL_STATUS_CODE = 0x16 AL_ERROR_FLAG = 1 AL_STATE = STATE_INIT STOP_MBX_HANDLER AL Status Changed.req (AL state, AL staus code)	INIT
56.2	BOOT	**SM_Chg** /not SM_SETTINGS_0_AND_1_MATCH => ignore	BOOT
57	BOOT	**AL_State Change.req (AL Status, AL Status Code)** /AL_STATE = STATE_INIT and AL_ERROR_FLAG = 1 => STOP_MBX_HANDLER AL_STATE = STATE_INIT AL_ERROR_FLAG = TRUE AL Status Changed.req (AL state, AL status code)	INIT
58	BOOT	**AL_State Change.req (AL Status, AL Status Code)** /((AL_STATE = STATE_INIT and AL_ERROR_FLAG = 0) or AL_STATE = STATE_PREOP or STATE_SAFEOP or STATE_OP or STATE_BOOT or unknown => ignore	BOOT

6.4.2 Mailbox handler state machine

6.4.2.1 Description

The mailbox handler is responsible for the coordination of master and slave regarding mailbox operation. Mailbox writes are forwarded to the specific machines and responses will be put to the read mailbox.

As there is no specific state orientated service, there is no state table.

The tasks or the mailbox handler are:

mapping of write mailbox services to protocol handler

queuing of service request to read mailbox

confirming transmittal of read mailbox.

The Protocol handler can be:

CoE state machine

EoE state machine

FoE state machine

profile specific state machine.

6.4.2.2 Primitives exchanged between DL, Mailbox handler and Protocol Handler

Table 104 shows the service primitives including their associated parameters issued by the Mailbox handler and received by the DL.

Table 104 – Primitives issued by Mailbox handler to DL

Primitive name	Associated parameters	Functions
Mailbox Read Upd.req	Length Address Channel Priority Type Cnt Service Data	Refer to Service Definition ETG.1000.3

Table 105 shows the service primitives including their associated parameters issued by the DL received by the Mailbox handler.

Table 105 – Primitives issued by DL to Mailbox handler

Primitive name	Associated parameters	Functions
Mailbox Write.ind	Length Address Channel Priority Type Cnt Service Data	Refer to Service Definition ETG.1000.3
Mailbox Read Upd.cnf	success	Refer to Service Definition ETG.1000.3

Table 106 shows the service primitives including their associated parameters issued by the Protocol handler and received by the Mailbox handler. The TYPE prefix is derived from type parameter of the corresponding DL service primitive.

Table 106 – Primitives issued by Protocol handler to Mailbox handler

Primitive name	Associated parameters	Functions
TYPE Mailbox Read Upd.req	Length Address Channel Priority Service Data	Refer to Mailbox Read Upd Service Definition ETG.1000.3

Table 107 shows the service primitives including their associated parameters issued by the Mailbox handler received by the Protocol handler. The TYPE prefix is derived from type parameter of the corresponding DL service primitive.

Table 107 – Primitives issued by Mailbox handler to Protocol handler

Primitive name	Associated parameters	Functions
TYPE Mailbox Write.ind	Length Address Channel Priority Service Data	Refer to Service Definition ETG.1000.3
TYPE Mailbox Read Upd.cnf	success	Refer to Mailbox Read Upd Service Definition ETG.1000.3

6.4.3 CoE state machine

6.4.3.1 Description

The CoE state machine is responsible for processing CoE services.

The main task is to execute the service requests and provide the response either

as single frame

or as sequence of frames (info services)

or as single frame with more follows indication

or as abort for erroneous termination of the service.

6.4.3.2 Primitive definitions

6.4.3.2.1 Primitives exchanged between Mailbox Handler and CoESM

The primitives between CoESM and Mailbox Handler are described in 6.4.2.2.

6.4.3.2.2 Primitives exchanged between Application and CoESM

Table 108 shows the service primitives including their associated parameters issued by the AL and received by the CoESM. The SDO Info services respresents a group of services (Get OD list, Get object description, Get entry description) that are distinguished by the opcode parameter.

Table 108 – Primitives issued by Application to CoESM

Primitive name	Associated parameters	Functions
SDO Download Expedited.rsp	Success Address Index Subindex	Refer to CoE Service Definition ETG.1000.5
SDO Download Normal.rsp	Success Address Index Subindex	Refer to CoE Service Definition ETG.1000.5
Download SDO Segment.rsp	Success Address Toggle	Refer to CoE Service Definition ETG.1000.5
SDO Upload Expedited.rsp	Success Address Index Subindex Size Data	Refer to CoE Service Definition ETG.1000.5
SDO Upload Normal.rsp	Success Address Index Subindex Complete Size Size Data	Refer to CoE Service Definition ETG.1000.5
Upload SDO Segment.rsp	Success Address Toggle More Follows Size Data	Refer to CoE Service Definition ETG.1000.5
Abort SDO Transfer.req	Address Index Subindex Reason	Refer to CoE Service Definition ETG.1000.5
SDO Info Service.rsp	Address Opcode Incomplete Fragments Left Size Data	Refer to CoE Service Definition ETG.1000.5
SDO Info Seg Service.req	Address Opcode Incomplete Fragments Left Size Data	Refer to CoE Service Definition ETG.1000.5
Emergency Service.req	Address Error Code Error Register Size Data	Refer to CoE Service Definition ETG.1000.5

Table 109 shows the service primitives including their associated parameters issued by the CoESM received by the AL.

Table 109 – Primitives issued by CoESM to Application

Primitive name	Associated parameters	Functions
SDO Download Expedited.ind	Address Index Subindex Size Data	Refer to CoE Service Definition ETG.1000.5
SDO Download Normal.ind	Address Index Subindex Complete Size Size Data	Refer to CoE Service Definition ETG.1000.5
Download SDO Segment.ind	Address Toggle More Follows Size Data	Refer to CoE Service Definition ETG.1000.5
SDO Upload Expedited.ind	Address Index Subindex	Refer to CoE Service Definition ETG.1000.5
SDO Upload Normal.ind	Address Index Subindex	Refer to CoE Service Definition ETG.1000.5
Upload SDO Segment.ind	Address Toggle	Refer to CoE Service Definition ETG.1000.5
Abort SDO Transfer.req	Address Index Subindex Reason	Refer to CoE Service Definition ETG.1000.5
SDO Info Service.ind	Address Opcode Incomplete Fragments Left Size Data	Refer to CoE Service Definition ETG.1000.5

6.4.3.2.3 Parameters of primitives

The parameters used with the primitives exchanged between CoESM and Application are described in ETG.1000.5.

6.4.3.3 CoESM State Table

Table 110 contains the complete description of the CoESM state machine.

Table 110 – CoESM state table

#	Current State	Event /Condition =>Action	Next State
1	OFF	**START MAILBOX** => Seg = 0	IDLE
2	OFF	**STOP MAILBOX** =>	OFF
3	IDLE	**START MAILBOX** =>	IDLE
4	IDLE	**STOP MAILBOX** =>	OFF

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State
5	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.Service != 2 && Service_Data.Service != 8 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
6	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.Service == 2 && Service_Data.Command_Specifier > 3 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
7	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 1 && Service_Data.Transfer_Type == 1 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
8	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 1 && Service_Data.Transfer_Type == 1 => Size = 4 - Service_Data.Data_Set_Size Index = Service_Data.Index SubIndex = Service_Data.SubIndex Data = Service_Data.Data SDO Download Expedited.ind (Address, Index, Subindex , Size, Data)	DWLE
9	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length =< 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 1 && Service_Data.Transfer_Type == 0 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR

10	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)**
		/Length > 10 && Service_Data.Service == 2 &&
		Service_Data.Command_Specifier == 1 && Service_Data.Transfer_Type == 0
		=>
		Seg = (Service Data.Complete_Size - (Length - 10))
		Toggle = FALSE
		Size = Length - 10
		Complete Size = Service Data.Complete_Size
		Index = Service_Data.Index
		SubIndex = Service_Data.SubIndex
		Data = Service_Data.Data
		
		SDO Download Normal.ind (Address, Index, Subindex, Complete Size, Size, Data)
			DWLN

11	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length < 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 0 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State
12	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length >=10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 0 => Length = 10 Service_Data.Service = 2 Service_Data.Command_Specifier = 4 Service_Data.Abort_Code = ABT_SEQ CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
13	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 2 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
14	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 2 => Index = Service_Data.Index SubIndex = Service_Data.SubIndex SDO Upload Expedited.ind (Address, Index, Subindex)	UPLE
15	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 2 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
16	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 2 => Index = Service_Data.Index SubIndex = Service_Data.SubIndex SDO Upload Normal.ind (Address, Index, Subindex)	UPLN
17	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 3 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 3 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
18	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 3 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 3 => Length = 10 Service_Data.Service = 2 Service_Data.Command_Specifier = 4 Service_Data.Abort_Code = ABT_SEQ CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State
19	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 8 && Service_Data.Service == 8 && Service_Data.Opcode == 1,3 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR

20	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)**
		/Length == 8 && Service_Data.Service == 8 && Service_Data.Opcode == 1,3
		=>
		Size = Length -6
		Opcode = Service_Data.Opcode
		Incomplete = Service_Data.Incomplete
		FragmentsLeft = Service_Data.FragmentsLeft
		
		SDO Info Service.ind (Address, Opcode, Incomplete, Fragments Left, Size, Data)
			INF

21	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 10 && Service_Data.Service == 8 && Service_Data.Opcode == 5 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR

22	IDLE	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)**
		/Length == 10&& Service_Data.Service == 8 && Service_Data.Opcode == 5
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = ABT_SEQ
		
		SDO Info Service.ind (Address, Opcode, Incomplete, Fragments Left, Size, Data)
			INF

23	IDLE	**CoE Read Upd.cnf (success)** => ignore	IDLE
24	IDLE	**Emergency Service.req (Address, Error Code, Error Register, Size, Data)** => Length = 10 Service_Data.Service = 1 Service_Data.Error Code = Error Code Service_Data.Error Register = Error Register Service_Data.Data = Data CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
25	IDLE	**other application service primitives** => ignore	IDLE
26	DWLE	**START MAILBOX** => ignore	DWLE
27	DWLE	**STOP MAILBOX** => CANCEL SERVICE	OFF

28	DWLE	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State

29	DWLE	**SDO Download Expedited.rsp (Success, Address, Index, Subindex)**
		=>
		Length = 6
		Service_Data.Service = 3
		Service_Data.Command_Specifier = 3
		Service_Data.Transfer Type = 0
		Service_Data.Index = Index
		Service_Data.Subindex = Subindex
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

30	DWLE	**other application service primitives** => ignore	DWLE
31	DWLN	**START MAILBOX** => ignore	DWLN
32	DWLN	**STOP MAILBOX** => CANCEL SERVICE	OFF

33	DWLN	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		Seg = 0
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

34	DWLN	**SDO Download Normal.rsp (Success, Address, Index, Subindex)**
		=>
		Length = 6
		Service_Data.Service = 3
		Service_Data.Command_Specifier = 3
		Service_Data.Transfer Type = 0
		Service_Data.Index = Index
		Service_Data.Subindex = Subindex
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

35	DWLN	**other application service primitives** => ignore	DWLN
36	DWLS	**START MAILBOX** => ignore	DWLS
37	DWLS	**STOP MAILBOX** => CANCEL SERVICE	OFF

38	DWLS	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		Seg = 0
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State
39	DWLS	**Download SDO Segment.rsp (Success, Address, Toggle)** => Length = 6 Service_Data.Service = 3 Service_Data.Command_Specifier = 1 Service_Data.Toggle, Toggle = !Toggle CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
40	DWLS	**other application service primitives** => ignore	DWLS
41	UPLE	**START MAILBOX** => ignore	UPLE
42	UPLE	**STOP MAILBOX** => CANCEL SERVICE	OFF

43	UPLE	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

44	UPLE	**SDO Upload Expedited.rsp (Success, Address, Index, Subindex, Size, Data)**
		=>
		Length = 10
		Service_Data.Service = 3
		Service_Data.Command_Specifier = 2
		Service_Data.Transfer Type = 1
		Service_Data.Data Set Size = 4 - Size
		Service_Data.Index = Index
		Service_Data.Subindex = Subindex
		Service_Data.Data = Data
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

45	UPLE	**other application service primitives** => ignore	UPLE
46	UPLN	**START MAILBOX** => ignore	UPLN
47	UPLN	**STOP MAILBOX** => CANCEL SERVICE	OFF

48	UPLN	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State

49	UPLN	**SDO Upload Normal.rsp (Success, Address, Index, Subindex, Complete Size, Size, Data)**
		=>
		Seg = Size - Complete Size
		Toggle = FALSE
		Length = 10 + Size
		Service_Data.Service = 3
		Service_Data.Command_Specifier = 2
		Service_Data.Transfer Type = 0
		Service_Data.Index = Index
		Service_Data.Subindex = Subindex
		Service_Data.Data = Data
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

50	UPLN	**other application service primitives** => ignore	UPLN
51	UPLS	**START MAILBOX** => ignore	UPLS
52	UPLS	**STOP MAILBOX** => CANCEL SERVICE	OFF

53	UPLS	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		Seg = 0
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

54	UPLS	**Upload SDO Segment.rsp (Success, Address, Toggle, More Follows, Size, Data)**
		=>
		Seg = (Seg -Size)
		if (Size > 7) then Service_Data.SegDataSize = 0 else
		Service_Data.SegDataSize =7-Size
		Length = Size + 3 + Service_Data.SegDataSize
		Service_Data.Service = 3
		Service_Data.Command_Specifier = 0
		Service_Data.Toggle = Toggle
		Service_Data.Data = Data
		Service_Data.More Follows = More Follows
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

55	UPLS	**other application service primitives** => ignore	UPLS
56	INF	**START MAILBOX** => ignore	INF
57	INF	**STOP MAILBOX** => CANCEL SERVICE	OFF

58	INF	**Abort SDO Transfer.req (Address, Index, Subindex, Reason)**
		=>
		Length = 10
		Service_Data.Service = 2
		Service_Data.Command_Specifier = 4
		Service_Data.Abort_Code = Reason
		Seg = 0
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State

59	INF	**SDO Info Service.rsp (Address, Opcode, Incomplete, Fragments Left, Size, Data)**
		=>
		if FragmentsLeft >0) then Seg = 0x80000000 else Seg = 0
		Length = Size + 6
		Service_Data.Service = 8
		Service_Data.Opcode = Opcode
		Service_Data. Incomplete = Incomplete
		Service_Data.Data = Data
		Service_Data. Fragments Left = Fragments Left
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

60	INF	**SDO Info Seg Service.req (Address, Opcode, Incomplete, Fragments Left, Size, Data)**
		=>
		if FragmentsLeft >0) then Seg = 0x80000000 else Seg = 0
		Length = Size + 6
		Service_Data.Service = 8
		Service_Data.Opcode = Opcode
		Service_Data. Incomplete = Incomplete
		Service_Data.Data = Data
		Service_Data. Fragments Left = Fragments Left
		
		CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

61	WUPD	**START MAILBOX** =>	WUPD
62	WUPD	**STOP MAILBOX** => RESET MAILBOX	OFF
63	WUPD	**CoE Mailbox Read Upd.cnf (success)** /Seg == 0 =>	IDLE
64	WUPD	**CoE Mailbox Read Upd.cnf (success)** /Seg == 0x80000000 && (Fragments Left == 0) => Seg = 0	IDLE
65	WUPD	**CoE Mailbox Read Upd.cnf (success)** /Seg == 0x80000000 && (Fragments Left > 0) =>	INF
66	WUPD	**CoE Mailbox Read Upd.cnf (success)** /Seg >0 && Seg < 0x80000000 =>	WSEG
67	WUPD	**CoE Mailbox Read Upd.cnf (success)** /Seg < 0 && Seg > 0x80000000 =>	WSEG
69	ERR	**START MAILBOX** =>	ERR
70	ERR	**STOP MAILBOX** => RESET MAILBOX	OFF
71	ERR	**ERR Mailbox Read Upd.cnf (success)** =>	IDLE
73	WSEG	**START MAILBOX** =>	WSEG
74	WSEG	**STOP MAILBOX** =>	OFF

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State
75	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.Service != 2 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER Seg =0 ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
76	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.Service == 2 && Service_Data.Command_Specifier != 0,3 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER Seg =0 ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
77	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length < 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 0 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE Seg = 0 ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
78	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length >= 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 0 && Seg < (Length - 3 - Service_Data.SegDataSize) => Length = 10 Service_Data.Service = 2 Service_Data.Command_Specifier = 4 Service_Data.Abort_Code = ABT_SEQ CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
79	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length >= 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 0 && Seg >= (Length - 3 - Service_Data.SegDataSize) && (Service_Data.More Follows != (0 < (Seg -(Length - 3 - Service_Data.SegDataSize))) || Toggle = Service_Data.Toggle) => Length = 10 Service_Data.Service = 2 Service_Data.Command_Specifier = 4 Service_Data.Abort_Code = ABT_SEQ CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
80	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length >= 10 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 0 && Seg >= (Length - 3 - Service_Data.SegDataSize) && Service_Data.More Follows == (0 < (Seg -(Length - 3 - Service_Data.SegDataSize))) && !Toggle = Service_Data.Toggle => Seg = (Seg -(Length - 3 - Service_Data.SegDataSize)) Size = Length - 3 - Service_Data.SegDataSize Toggle = Service_Data.Toggle More Follows = Service_Data.More Follows Data = Service_Data.Data Download SDO Segment.ind (Address, Toggle, More Follows, Size, Data)	DWLS

Table 110 *(continued)*

#	Current State	Event /Condition =>Action	Next State
81	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length != 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 3 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
82	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 3 && Toggle = Service_Data.Toggle => Length = 10 Service_Data.Service = 2 Service_Data.Command_Specifier = 4 Service_Data.Abort_Code = ABT_SEQ CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
83	WSEG	**CoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 6 && Service_Data.Service == 2 && Service_Data.Command_Specifier == 3 && !Toggle = Service_Data.Toggle => Toggle = Service_Data.Toggle Upload SDO Segment.ind (Address, Toggle)	UPLS
84	WSEG	**Emergency Service.req (Address, Error Code, Error Register, Size, Data)** => Length = 10 Service_Data.Service = 1 Service_Data.Error Code = Error Code Service_Data.Error Register = Error Register Service_Data.Data = Data CoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
85	WSEG	**other application service primitives** => ignore	IDLE

6.4.4 EoE state machine

6.4.4.1 Description

The EoE state machine is responsible for conveying standard Ethernet frames over EtherCAT. The state machine receives and transmits EtherCAT PDUs either with the complete Ethernet frame or a fragment of the Ethernet frame, which can be combined to a complete frame. After the reassembly (which is out of scope of this machine) the resulting Ethernet frame can be treated as if transmitted with Ethernet without EtherCAT services.

The ingress and egress rules of an Ethernet port that may be associated with the EoE services are specified in IEEE 802.1D.

General Time Stamp definitions:

- 32 bit, 1 ns resolution
- DC System Time can be used
- Time Stamp trigger is the beginning of the destination address (DA)

Time Stamp appended (TA = 1):

- Slave -> master: Time Stamp contains exact receive time

- Master -> slave: Time Stamp contains desired send time
- Slave should always append a Time Stamp if it has this feature
- Time Stamp extends frame data by 32 bit
- TA bit allowed in last fragment only (LF=1)
- If Time Stamp does not fit into the last fragment -> add a fragment
    - fill the “last” fragment with parts of the Time Stamp (LF=0, TA=0) and send a very last fragment with the rest of the Time Stamp (LF=1, TA=1)

Time Stamp requested (TR = 1):

- Response with the exact send time and the same FrameNo requested
- Response should be send as soon as possible

6.4.4.2 Primitive definitions

6.4.4.2.1 Primitives exchanged between Mailbox Handler and EoESM

The primitives between EoESM and Mailbox Handler are described in 6.4.2.2.

6.4.4.2.2 Primitives exchanged between Application and EoESM

Table 111 shows the service primitives including their associated parameters issued by the AL and received by the EoESM.

Table 111 – Primitives issued by Application to EoESM

Primitive name	Associated parameters	Functions
Initiate_EoE.req	Address, Port Time Appended Time Requested Frame Number Complete Size Last Fragment Size Data Time Stamp	Refer to CoE Service Definition ETG.1000.5
Initiate_EoE.rsp	Address, Frame Number Time Stamp	Refer to CoE Service Definition ETG.1000.5
EoE Fragment.req	Address, Port Time Appended Frame Number Offset Last Fragment Size Data Time Stamp	Refer to CoE Service Definition ETG.1000.5
Set IP Parameter.rsp	Address, Reason	Refer to CoE Service Definition ETG.1000.5
Set Address Filter.rsp	Address, Reason	Refer to CoE Service Definition ETG.1000.5

Table 112 shows the service primitives including their associated parameters issued by the EoESM received by the AL.

Table 112 – Primitives issued by EoESM to Application

Primitive name	Associated parameters	Functions
Initiate_EoE.ind	Address, Port Time Appended Time Requested Frame Number Complete Size Last Fragment Size Data Time Stamp	Refer to CoE Service Definition ETG.1000.5
EoE Fragment.ind	Address, Port Time Appended Frame Number Offset Last Fragment Size Data Time Stamp	Refer to CoE Service Definition ETG.1000.5
Set IP Parameter.ind	Address, MAC Address IP Address Subnet Mask Default Gateway DNS Server DNS Name	Refer to CoE Service Definition ETG.1000.5
Set Address Filter.ind	Address, Broadcast Fowarding MAC Address Filters MAC Filter Masks	Refer to CoE Service Definition ETG.1000.5

6.4.4.2.3 Parameters of primitives

The parameters used with the primitives exchanged between EoESM and Application are described in ETG.1000.5.

6.4.4.3 EoESM State Table

Table 113 contains the complete description of the EoESM state machine. The Set IP Parameter and Set Address Filter are handled in any state in such a way that the requested parameter will be changed if possible.

NOTE: SSeg is a local variable that calculates the number of octets of an Ethernet frame.

Table 113 – EoESM state table

#	Current State	Event /Condition =>Action	Next State
1	OFF	**START MAILBOX** => SSeg = 0	IDLE
2	OFF	**STOP MAILBOX** =>	OFF
3	IDLE	**START MAILBOX** =>	IDLE
4	IDLE	**STOP MAILBOX** => Terminate Segmented Services	OFF
5	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg == 0 && Length < 36 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
6	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg == 0 && Length >= 36 && (Service_Data.Fragment > 0 || Service_Data.CompleteSize < 2) => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
7	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg == 0 && Length >= 36 && Service_Data.Fragment == 0 && Service_Data.CompleteSize >= 2 && Service_Data.LastFragment == 1 => TimeAppended = Service_Data.TimeAppended TimeRequested= Service_Data.TimeRequested Size = Length-4-TimeAppended*4 if (TimeAppend) Timestamp = Service_Data[Length -4..Length-1] Frame Number = Service_Data.Frame Number Complete Size = Service_Data.CompleteSize Fragment Number = Service_Data.Fragment LastFragment = 1 Data = Service_Data.EoE Data Initiate_EoE.ind (Address, Port, Time Appended, Time Requested, Frame Number, Complete Size, Last Fragment, Size, Data,Timestamp)	IDLE
8	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg == 0 && Length >= 36 && Service_Data.Fragment == 0 && Service_Data.CompleteSize >= 2 && Service_Data.LastFragment == 0 && ((Length -4) mod 32) == 0 => TimeAppended = Service_Data.TimeAppended TimeRequested= Service_Data.TimeRequested Size,SSeg = Length-4 Frame Number = Service_Data.Frame Number Fragment Number = Service_Data.Fragment Complete Size = Service_Data.CompleteSize LastFragment = 0 Data = Service_Data.EoE Data Initiate_EoE.req (Address, Port, Time Appended, Time Requested, Frame Number, Complete Size, Last Fragment, Size, Data,Timestamp)	IDLE

Table 113 *(continued)*

#	Current State	Event /Condition =>Action	Next State
9	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg == 0 && Length >= 36 && Service_Data.Fragment == 0 && Service_Data.CompleteSize >= 2 && Service_Data.LastFragment == 0 && ((Length -4) mod 32) != 0 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
10	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg != 0 && (Service_Data.Fragment == 0 || Service_Data.Offset*32 != SSeg) => SSeg = 0 Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR

11	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)**
		/ Service_Data.FrameType == 0 && SSeg != 0 && Service_Data.Fragment != 0 && Service_Data.Offset*32 == SSeg && Service_Data.LastFragment == 1
		=>
		SSeg = 0
		TimeAppended = Service_Data.TimeAppended
		Size = Length-4-TimeAppended*4
		if (TimeAppend) Timestamp = Service_Data[Length -4..Length-1]
		Frame Number = Service_Data.Frame Number
		Offset = Service_Data.Offset
		Fragment Number = Service_Data.Fragment
		LastFragment = 1
		Data = Service_Data.EoE Data
		
		EoE_Fragment.ind (Address, Port, Time Appended, Frame Number, Offset, Last Fragment, Size, Data, Timestamp)
			IDLE

12	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg != 0 && Service_Data.Fragment != 0 && Service_Data.Offset*32 == SSeg && Service_Data.LastFragment == 0 && ((Length -4) mod 32) == 0 => SSeg = SSEG + Length-4 Frame Number = Service_Data.Frame Number Offset = Service_Data.Offset Fragment Number = Service_Data.Fragment LastFragment = 0 Data = Service_Data.EoE Data EoE_Fragment.ind (Address, Port, Time Appended, Frame Number, Offset, Last Fragment, Size, Data, Timestamp)	IDLE
13	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** / Service_Data.FrameType == 0 && SSeg != 0 && Service_Data.Fragment != 0 && Service_Data.Offset*32 == SSeg && Service_Data.LastFragment == 0 && ((Length -2) mod 32) != 0 => SSeg = 0 Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR

Table 113 *(continued)*

#	Current State	Event /Condition =>Action	Next State
14	IDLE	**Initiate_EoE.req (Address, Port, Time Appended, Time Requested, Frame Number, Complete Size, Last Fragment, Size, Data,Timestamp)** => Service_Data.TimeAppended = TimeAppended Service_Data.TimeRequested = TimeRequested Length = Size+4+TimeAppended*4 if (TimeAppend) Service_Data[Length -4..Length-1] = Timestamp Service_Data.Frame Number = Frame Number Service_Data.Fragment Number = Fragment Service_Data.Complete Size = CompleteSize Service_Data.LastFragment = LastFragment Service_Data.Port = Port Service_Data.EoE Data = Data EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD

15	IDLE	**EoE_Fragment.req(Address, Port, TimeAppended, Frame Number, Offset, Last Fragment, Size, Data,Timestamp)**
		=>
		Service_Data.TimeAppended = TimeAppended
		Length = Size+4+TimeAppended*4
		if (TimeAppend) Service_Data[Length -4..Length-1] = Timestamp
		Service_Data.Frame Number = Frame Number
		Service_Data.Fragment Number = Fragment
		Service_Data.Complete Size = CompleteSize
		Service_Data.LastFragment = LastFragment
		Service_Data.Port = Port
		Service_Data.EoE Data = Data
		
		EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

16	IDLE	**EOE Read Upd.cnf (success)** => ignore	IDLE
17	IDLE	**other application service primitives** => ignore	IDLE

18	IDLE	**Initiate_EoE.rsp(Address, Frame Number, Timestamp)**
		=>
		Service_Data.FrameType = 1
		Service_Data.TimeAppended = 1
		Service_Data.TimeStamp = Timestamp
		
		EOE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

19	WUPD	**START MAILBOX** =>	WUPD
20	WUPD	**STOP MAILBOX** => RESET MAILBOX	OFF
21	WUPD	**EOE Read Upd.cnf (success)** =>	WUPD
22	WUPD	**EOE Read.ind** =>	IDLE
23	WUPD	**Timeout** => RESET MAILBOX	OFF
24	ERR	**START MAILBOX**=>	ERR
25	ERR	**STOP MAILBOX** => RESET MAILBOX	OFF
26	ERR	**ERR Read Upd.cnf (success)** =>	ERR
27	ERR	**ERR Read.ind** =>	IDLE

Table 113 *(continued)*

#	Current State	Event /Condition =>Action	Next State
28	ERR	**Timeout** => RESET MAILBOX	OFF
29	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length > 8 (fits request) && Service_Data.FrameType == 2 => Extract IP Parameter from Service Data. IP Parameter Set IP Parameter.ind(Address, Mac Address, IP Adress, Subnet Mask, Default Gateway, DNS Server, DNS Name)	IPPAR

30	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)**
		/Length not (fits request) && Service_Data.FrameType == 2
		=>
		Length = 4
		Service_Data.FrameType = 3
		Service_Data.Result = 2
		Service_Data.Port = 0
		
		EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

31	IPPAR	**START MAILBOX** => ignore	IPPAR
32	IPPAR	**STOP MAILBOX** => CANCEL SERVICE	OFF

33	IPPAR	**Set IP Parameter.rsp(+) (Adress)**
		=>
		Length = 4
		Service_Data.FrameType = 3
		Service_Data.Result = 0
		Service_Data.Port = 0
		EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

34	IPPAR	**Set IP Parameter.rsp(-) (Adress, Reason)**
		=>
		Length = 4
		Service_Data.FrameType = 3
		Service_Data.Result = Reason
		Service_Data.Port = 0
		
		EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

35	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length > 6 (fits request) && Service_Data.FrameType == 4 => Extract Address Fileter from Service Data SetAddress Filter.ind(Address, Broadcast forwarding, MAC Address Filter 1..16, MAC Filter Mask 1..4)	MFLT
36	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length not (fits request) && Service_Data.FrameType == 4 => Length = 4 Service_Data.FrameType = 5 Service_Data.Result = 2 Service_Data.Port = 0 EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
37	MFLT	**START MAILBOX** => ignore	MFLT
38	MFLT	**STOP MAILBOX** => CANCEL SERVICE	OFF

Table 113 *(continued)*

#	Current State	Event /Condition =>Action	Next State

39	MFLT	**SetAddress Filter.rsp(+) (Adress)**
		=>
		Length = 4
		Service_Data.FrameType = 5
		Service_Data.Result = 0
		Service_Data.Port = 0
		
		EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

40	MFLT	**SetAddress Filter.rsp(-) (Adress, Reason)**
		=>
		Length = 4
		Service_Data.FrameType = 5
		Service_Data.Result = Reason
		Service_Data.Port = 0
		EoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

41	IDLE	**EoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.FrameType != 0,2,4 => ignore	IDLE

6.4.5 FoE state machine

6.4.5.1 Description

The FoE state machine is responsible for conveying Files over EtherCAT. The state machine receives and transmits EtherCAT PDUseither with transfer command (Write means load file to slave, Read means load file to master).

6.4.5.2 Primitive definitions

6.4.5.2.1 Primitives exchanged between Mailbox Handler and FoESM

The primitives between FoESM and Mailbox Handler are described in 6.4.2.2.

6.4.5.2.2 Primitives exchanged between Application and FoESM

Table 114 shows the service primitives including their associated parameters issued by the AL and received by the FoESM.

Table 114 – Primitives issued by Application to FoESM

Primitive name	Associated parameters	Functions
FoE Data.req	Address, Packet Number Size Data	Refer to FoE Service Definition ETG.1000.5
FoE Ack.req	Address, Packet Number	Refer to FoE Service Definition ETG.1000.5
FoE Error.req	Address, Error Code, Error Text	Refer to FoE Service Definition ETG.1000.5

Table 115 shows the service primitives including their associated parameters issued by the FoESM received by the AL.

Table 115 – Primitives issued by FoESM to Application

Primitive name	Associated parameters	Functions
FoE Write.ind	Address, Password File Name	Refer to FoE Service Definition ETG.1000.5
FoE Read.ind	Address, Password File Name	Refer to FoE Service Definition ETG.1000.5
FoE Data.ind	Address, Packet Number Size Data	Refer to FoE Service Definition ETG.1000.5
FoE Ack.ind	Address, Packet Number	Refer to FoE Service Definition ETG.1000.5
FoE Error.ind	Address, Error Code, Error Text	Refer to FoE Service Definition ETG.1000.5

6.4.5.2.3 Parameters of primitives

The parameters used with the primitives exchanged between FoESM and Application are described in ETG.1000.5.

6.4.5.3 FoESM State Table

Table 116 contains the complete description of the FoESM state machine.

Table 116 – FoESM state table

#	Current State	Event /Condition =>Action	Next State
1	OFF	**START MAILBOX** => Seg = 0	IDLE
2	OFF	**STOP MAILBOX** =>	OFF
3	IDLE	**START MAILBOX** =>	IDLE
4	IDLE	**STOP MAILBOX** =>	OFF
5	IDLE	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode == 0 || Service_Data.OpCode> 4 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
6	IDLE	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length =< 6 && Service_Data.OpCode == 2 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
7	IDLE	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length > 6 && Service_Data.OpCode == 2 => Password = Service_Data.Password Filename = Service_Data.FileName FoE Write.ind (Address, Password, Filename)	DWLS

#	Current State	Event /Condition =>Action	Next State
8	IDLE	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length =< 6 && Service_Data.OpCode == 1 => Length = 4 Service_Data == 1, MBXERR_INVALIDSIZE ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
9	IDLE	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length > 6 && Service_Data.OpCode == 1 => Password = Service_Data.Password Filename = Service_Data.FileName FoE Read.ind (Address, Password, Filename)	UPLS
10	IDLE	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Length == 3 && (Service_Data.OpCode == 3 || Service_Data.OpCode == 4) => Length = 6 Service_Data.OpCode = 5 Service_Data.Error_Code = ABT_SEQ FoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	WUPD
11	IDLE	**FoE Read Upd.cnf (success)** => ignore	IDLE
12	IDLE	**other application service primitives** => ignore	IDLE
13	DWLS	**START MAILBOX** => ignore	DWLS
14	DWLS	**STOP MAILBOX** => CANCEL SERVICE	OFF

Table 116 *(continued)*

#	Current State	Event /Condition =>Action	Next State

15	DWLS	**FoE Error.req (Address, ErrorCode, ErrorText)**
		=>
		Seg = 0
		Length = 6 + size(Error_Text)
		Service_Data.OpCode = 5
		Service_Data.Error_Code = Error_Code
		Service_Data.Error_Text= Error_Text
		FoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

16	DWLS	**FoE Ack.req (Address, Packet Number)**
		=>
		Length = 6
		Service_Data.OpCode = 4
		Service_Data.PacketNumber = Packet Number
		Seg = Packet Number + 1
		FoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			DWUPD

17	DWLS	**other application service primitives** => ignore	DWLS
18	UPLS	**START MAILBOX** => ignore	UPLS
19	UPLS	**STOP MAILBOX** => CANCEL SERVICE	OFF

Table 116 *(continued)*

#	Current State	Event /Condition =>Action	Next State

20	UPLS	**FoE Error.req (Address, ErrorCode, ErrorText)**
		=>
		Seg = 0
		Length = 6 + size(Error_Text)
		Service_Data.OpCode = 5
		Service_Data.Error_Code = Error_Code
		Service_Data.Error_Text= Error_Text
		FoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

11	IDLE	**FoE Read Upd.cnf (success)** => ignore	IDLE

21	UPLS	**FoE Data.req (Address, Packet Number, Size, Data)**
		/Size = FullSize
		=>
		Length = 6 + SIze
		Service_Data.OpCode = 3
		Service_Data.PacketNumber = Packet Number
		Service_Data.Data = Data
		Seg = Packet Number + 1
		FoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			UWUPD

22	UPLS	**FoE Data.req (Address, Packet Number, Size, Data)**
		/Size< FullSize
		=>
		Length = 6 + SIze
		Service_Data.OpCode = 3
		Service_Data.PacketNumber = Packet Number
		Service_Data.Data = Data
		Seg = Packet Number + 1
		FoE Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)
			WUPD

23	UPLS	**other application service primitives** => ignore	UPLS
24	UWUPD	**START MAILBOX** =>	UWUPD
25	UWUPD	**STOP MAILBOX** => RESET MAILBOX	OFF
26	UWUPD	**FoE Read Upd.cnf (success)** =>	UWSEG
27	UWUPD	**Timeout** => RESET MAILBOX	OFF
28	WUPD	**START MAILBOX** =>	WUPD
29	WUPD	**STOP MAILBOX** => RESET MAILBOX	OFF
30	WUPD	**FoE Read Upd.cnf (success)** =>	IDLE
31	WUPD	**Timeout** => RESET MAILBOX	OFF
32	DWUPD	**START MAILBOX** =>	DWUPD
33	DWUPD	**STOP MAILBOX** => RESET MAILBOX	OFF
34	DWUPD	**FoE Read Upd.cnf (success)** =>	DWSEG

Table 116 *(continued)*

#	Current State	Event /Condition =>Action	Next State
35	DWUPD	**Timeout** => RESET MAILBOX	OFF
36	ERR	**START MAILBOX** =>	ERR
37	ERR	**STOP MAILBOX** => RESET MAILBOX	OFF
38	ERR	**ERR Read Upd.cnf (success)** =>	IDLE
39	ERR	**Timeout** => RESET MAILBOX	OFF
40	DWSEG	**START MAILBOX** =>	DWSEG
41	DWSEG	**STOP MAILBOX** =>	OFF
42	DWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode != 3 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER Seg =0 FoE Error.ind (Address, ErrorCode = ABT_SEQ, ErrorText) ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
43	DWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode == 3 && Service_Data.PackeNumber != Seq => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER Seg =0 FoE Error.ind (Address, ErrorCode = ABT_SEQ, ErrorText) ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
44	DWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode == 3 && Service_Data.PackeNumber != Seq && Length = FullSize + 6 => Size = Length - 6 Packet Number = Service_Data.Packet Number Data = Service_Data.Data FoE Data.ind (Address, Packet Number, Size, Data)	DWLS
45	DWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode == 3 && Service_Data.PackeNumber != Seq && Length < FullSize + 6 => Size = Length - 6 Packet Number = Service_Data.Packet Number Data = Service_Data.Data FoE Data.ind (Address, Packet Number, Size, Data)	IDLE
46	DWSEG	**other application service primitives** => ignore	DWSEG
47	UWSEG	**START MAILBOX** =>	UWSEG
48	UWSEG	**STOP MAILBOX** =>	OFF

Table 116 *(continued)*

#	Current State	Event /Condition =>Action	Next State
49	UWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode != 4 => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER Seg =0 FoE Error.ind (Address, ErrorCode = ABT_SEQ, ErrorText) ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
50	UWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode == 4 && Service_Data.PackeNumber != Seq => Length = 4 Service_Data == 1, MBXERR_INVALIDHEADER Seg =0 FoE Error.ind (Address, ErrorCode = ABT_SEQ, ErrorText) ERR Mailbox Read Upd.req (Length, Address, Channel, Priority, Service_Data)	ERR
51	UWSEG	**FoE Mailbox Write.ind (Length, Address, Channel, Priority, Service Data)** /Service_Data.OpCode == 4 && Service_Data.PackeNumber == Seq => Packet Number = Service_Data.Packet Number FoE Ack.req (Address, Packet Number)	UPLS
52	UWSEG	**other application service primitives** => ignore	UWSEG

6.5 DLL mapping protocol machine (DMPM)

The services specified in ETG.1000.3 are directly used in the ARPMs.

Bibliography

IEC 61131-3, *Programmable controllers – Programming languages*

IEC/TR 61158-1 (Ed.2.0), *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-3-12, *Industrial communication networks – Fieldbus specifications - Part 3-12: data-link layer service definition – Type 12 elements*

IEC 61158-4-12, *Industrial communication networks – Fieldbus specifications - Part 4-12: data-link layer protocol specification – Type 12 elements*

IEC 61158-5-12, *Industrial communication networks – Fieldbus specifications - Part 5-12: Application layer service definition – Type 12 elements*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

ISO/IEC 646, *Information technology – ISO 7-bit coded character set for information interchange*

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

ISO/IEC 8859-1, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

ISO/IEC 10646, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

ISO 15745-1, *Industrial automation systems and integration – Open systems application integration framework – Part 1: Generic reference description*

IETF RFC 792, *Internet Control Message Protocol*; available at <http://www.ietf.org>

ETG.1000.2 *EtherCAT Specification Part 2: Physical layer specification and service definition*

ETG.1000.3 *EtherCAT Specification Part 3: Data Link Layer service definition*

ETG.1000.4 *EtherCAT Specification Part 4: Data-link layer protocol specification*

ETG.1000.5 *EtherCAT Specification Part 5: Application layer service definition*

ETG.1000.6 *EtherCAT Specification Part 6: Application layer protocol specification*

ETG.1100 *EtherCAT Specification Communication profiles*

____