IEC 61158-6-24

Edition 2.0 2023-03

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications –
Part 6-24: Application layer protocol specification – Type 24 elements

IEC 61158-6-24:2023-03(en)

IEC®

IEC 61158-6-24

Edition 2.0 2023-03

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications –
Part 6-24: Application layer protocol specification – Type 24 elements

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-6642-7

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-24: Application layer protocol specification – Type 24 elements

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in the IEC 61784-1 series and the IEC 61784-2 series.

IEC 61158-6-24 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation. It is an International Standard.

This second edition cancels and replaces the first edition published in 2014. This edition constitutes a technical revision.

The main changes with respect to the previous edition are listed below:

- addition of a new PDU type which called "Short PDU type II" in 4.2;
- update of Table 4;
- addition of examples of Synchronous Command communication in 9.2.1, Figure 27 and Figure 28.

The text of this International Standard is based on the following documents:

Draft	Report on voting
65C/1204/FDIS	65C/1245/RVD

Full information on the voting for its approval can be found in the report on voting indicated in the above table.

The language used for the development of this International Standard is English.

This document was drafted in accordance with ISO/IEC Directives, Part 2, and developed in accordance with ISO/IEC Directives, Part 1 and ISO/IEC Directives, IEC Supplement, available at www.iec.ch/members_experts/refdocs. The main document types developed by IEC are described in greater detail at www.iec.ch/publications.

A list of all parts of the IEC 61158 series, published under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC web site.

The committee has decided that the contents of this document will remain unchanged until the stability date indicated on the IEC website under webstore.iec.ch in the data related to the specific document. At this date, the document will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

INTRODUCTION

This document is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the "three-layer" fieldbus reference model described in IEC 61158-1.

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower layer. The primary aim of this document is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer application entities (AEs) at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

- as a guide for implementers and designers;
- for use in the testing and procurement of equipment;
- as part of an agreement for the admittance of systems into the open systems environment;
- as a refinement to the understanding of time-critical communications within OSI.

This document is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this document together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems can work together in any combination.

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-24: Application layer protocol specification – Type 24 elements

1 Scope

1.1 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 part of IEC 61158 provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 24 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 document defines in an abstract way the externally visible behavior provided by the Type 24 fieldbus application layer in terms of

- the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,
- the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,
- the application context state machines defining the application service behavior visibly between communicating application entities, and
- the application relationship state machines defining the communication behavior visibly between communicating application entities.

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

- define the representation-on-wire of the service primitives defined in IEC 61158-5-24, and
- define the externally visible behavior associated with their transfer.

This document specifies the protocol of the Type 24 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI Application Layer Structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this document is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-24.

A secondary objective is to provide migration paths from previously existing industrial communications protocols. It is this latter objective which gives rise to the diversity of protocols standardized in the IEC 61158-6 series.

1.3 Conformance

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

Conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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.

NOTE All parts of the IEC 61158 series, as well as the IEC 61784-1 series and the IEC 61784-2 series are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61158-5-24:2023, *Industrial communication networks – Fieldbus specifications – Part 5-24: Application layer service definition – Type 24 elements*

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

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

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

ISO/IEC 9899, *Information technology – Programming languages – C*

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

ISO/IEC 19501:2005, *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

ISO/IEC/IEEE 60559:2020, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

3 Terms, definitions, symbols, abbreviated terms, and conventions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at https://www.electropedia.org/
- ISO Online browsing platform: available at https://www.iso.org/obp

3.1 Referenced terms and definitions

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

3.1.1 Terms and definitions from ISO/IEC 7498-1

For the purposes of this document, the following terms as defined in ISO/IEC 7498-1 apply:

a) abstract syntax;
b) application-entity;
c) application process;
d) application protocol data unit;
e) application-process-invocation;
f) (N)-facility;
g) (N)-function;
h) peer-(N)-entities;
i) presentation context;
j) real system;
k) transfer syntax.

3.1.2 Terms and definitions from ISO/IEC 9545

For the purposes of this document, the following terms as defined in ISO/IEC 9545 apply:

a) application-association;
b) application-context;
c) application-entity-invocation;
d) application-entity-type;
e) application-service-element.

3.1.3 Terms and definitions from ISO/IEC 8824-1

For the purposes of this document, the following terms as defined in ISO/IEC 8824-1 apply:

a) simple type;
b) component;
c) component type;
d) integer type;
e) bitstring type;
f) octetstring type;
g) null type;
h) sequence type;
i) sequence of type;
j) choice type;
k) IA5String type;
l) encoding.

3.1.4 Terms and definitions from ISO/IEC 10731

For the purposes of this document, the following terms as defined in ISO/IEC 10731 apply:

a) OSI-service-primitive; primitive;
b) OSI-service-provider; provider;
c) OSI-service-user; user.

3.1.5 Terms and definitions from ISO/IEC 19501

For the purposes of this document, the following terms as defined in ISO/IEC 19501 apply:

a) event;
b) state;
c) state machine;
d) substate;
e) submachine;
f) transition.

3.2 Additional terms and definitions

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

3.2.1
alarm
field device status to tell that the device has detected a fatal problem to be solved and cannot continue normal working, through the field device control (FDC) service of the Type 24 fieldbus

Note 1 to entry: Any alarm statuses are latched and need some operations to be cleared.

Note 2 to entry: Alarms are classified into three groups; communication alarms, illegal-command-related ones, and application specific ones. But concrete definitions are dependent on implementation of each field devices.

3.2.2
application process object
network representation of a specific aspect of an application process (AP), which is modelled as a network accessible object contained within an AP or within another APO

Note 1 to entry: Refer to IEC 61158-1, 9.3.4.

3.2.3
application process context
AP context
shared knowledge or a common set of rules, governing communication of FAL application entities (AEs) and describing the permissible collective communications behavior between the AEs that are party to a specific set of application relationships (ARs)

Note 1 to entry: Data within AP context can be specified by the user in advance, by the option selected while the user uses a field bus management (FSM) service to read out the facility of peer AP, by the automatic negotiation function that the FSM system handles, and so on. The method that is to be adopted depends on the specification of each implementation.

3.2.4
application process type
AP type
description of a classification of application processes (APs) in terms of a set of capabilities for FAL of the Type 24 fieldbus

Note 1 to entry: AP types are classified into three, C1 master AP, C2 master AP and slave AP, by their application roles in the fieldbus network.

3.2.5
async command
type of a command application protocol data unit (APDU) of the FDC service of the Type 24 FAL, which can be issued any time after the previous transaction without consideration of synchronization with the communication cycle

Note 1 to entry: Definitions, which command should be async one or not, are dependent on an application. They can be provided as a registered set of commands and responses or device profiles, see 4.4 and Annex A.

3.2.6
asynchronous communication
state or a way of communication for the FDC service of the Type 24 FAL, in which a command can be issued any time after the previous transaction without consideration of synchronization with the communication cycle

Note 1 to entry: In this state, sync commands cannot be issued, but async commands can.

3.2.7
attribute
information or parameter contained in variable portions of an object

Note 1 to entry: Typically, they provide status information or govern the operation of an object. Attributes also affect the behavior of an object.

3.2.8
C1 master
AP type that has master facilities for the FDC service of the Type 24 FAL, or the device implementing that AP type

Note 1 to entry: Only one C1 master exists in a network of the Type 24 fieldbus.

3.2.9
C2 master
AP type that has only monitor facilities for the FDC service but requester facilities for message (MSG) service of the Type 24 FAL, or the device implementing that AP type

Note 1 to entry: Less than two C2 masters can exist in a network of the Type 24 fieldbus.

3.2.10
command
PDU issued by a requester or a master to make a responder or a slave execute some functions

3.2.11
communication
process to exchange information in a formal manner between two or more devices, users, APs or entities

3.2.12
transfer
process to convey a PDU from a sender to a receiver

3.2.13
transmission
process to send out and propagate electrical signals or encoded data

3.2.14
communication cycle
period of repetitive activities synchronized with the transmission cycle while the connection establishing for the FDC protocol of the Type 24 FAL

Note 1 to entry: Communication cycle can synchronize with the transmission cycle multiplied by a specified scaling factor.

3.2.15
connection
context or logical binding under specific conditions for the FDC protocol between a master object and a slave object for the Type 24 FAL

3.2.16
cyclic
repetitive in a regular manner

3.2.17
cyclic communication
transmission mode in which request PDUs and response PDUs are exchanged repetitively in the scheduled time slots synchronized with a transmission cycle for the lower layer protocol of the Type 24

Note 1 to entry: In the AL, the communication cycle arises from the transmission cycle in this mode.

3.2.18
cycle scale counter
counter to generate a communication cycle by means of scaling a primary cycle or a transmission cycle

3.2.19
device ID
part of "Device Information" to identify the device for a specific product type or model of the Type 24 fieldbus

3.2.20
device information
formatted and device-embedded information to characterize a device, which mainly consists of data for device model identification and device-profile specific parameters for the Type 24 fieldbus

3.2.21
device profile
collection of device-model-common information and functionality providing consistency between different device models among the same kind of devices

3.2.22
dual transfer
transfer mode for the FDC protocol of the Type 24 FAL, in which a sender sends a same PDU twice a transaction and a receiver uses them to detect and recover a communication error such as data-corruption or data-loss in cyclic communication mode

3.2.23
event driven communication
transmission mode for the lower layer protocol of the Type 24 fieldbus in which a transaction of command-response-exchanging arises as user's demands

Note 1 to entry: Both the transmission cycle and the communication cycle don't arise in this mode.

3.2.24
error
abnormal condition or malfunction for communication or any other activities

3.2.25
field device control
FDC service
time-critical communication service that handles a fixed length command data to control a field device and the corresponding feedback response data in a severe restriction on delay or jitter for the communication timing for the Type 24 FAL

3.2.26
field device protocol
FDC protocol
time-critical communication protocol that handles a fixed length command data to control a field device and the corresponding feedback response data in a severe restriction on delay or jitter for the communication timing for the Type 24 FAL

3.2.27
master
class or its instance object of FDC application service element (ASE) who plays a role of a command requester for the Type 24 FAL

3.2.28
message service
MSG service
communication service that handles the variable length data and not required a severe restriction on response time

3.2.29
monitor
class or its instance object of FDC ASE who plays a role of a watcher or subscriber of commands and response between other communication nodes for the Type 24 FAL

3.2.30
monitor slave
variant of slave AP type who has both slave class and monitor class for FDC ASE of the Type 24 FAL

3.2.31
network clock
synchronized and periodically running counter that each nodes in a same network have, which becomes an oscillation source of the transmission cycle

3.2.32
primary cycle
period of repetitive activities synchronized with the transmission cycle before the connection establishing for the FDC protocol in Type 24 FAL

3.2.33
protocol machine
state machine that realizes a protocol as the main function of the entity in each layer

3.2.34
requester
class or its instance object of MSG ASE who plays a role of a command requester or sender for the Type 24 FAL

3.2.35
responder
class or its instance object of MSG ASE who plays a role of a command responder or receiver for the Type 24 FAL

3.2.36
response
PDU issued by a responder or a slave to inform a result or some status for the received command to a requester or a master

3.2.37
service
operation or process that an object performs upon request from another object

3.2.38
single transfer
normal transfer mode for the FDC protocol of the Type 24 FAL in which a sender sends a PDU once a transaction

3.2.39
slave AP
AP type that has slave facilities for the FDC service of the Type 24 FAL, or the device implementing that AP type

3.2.40
slave
class or its instance object of FDC ASE who plays a role of a responder for the Type 24 FAL

3.2.41
sync command
type of command APDU of the FDC service of the Type 24 FAL, which is issued at the synchronized timing with every communication cycle

Note 1 to entry: The definitions, which command is sync one or not, are dependent on an application. They can be provided as a registered set of commands and responses or device profiles, see 4.4 and Annex A.

3.2.42
synchronous communication
state or a way of communication for the FDC service of the Type 24 FAL, in which a command is issued at the synchronized timing with every communication cycle

Note 1 to entry: In this state, both sync commands and async ones can be issued.

Note 2 to entry: In this state, an out-of-synchronization error of APs shall be detected by measures of the watchdog counter.

3.2.43
transmission cycle
period of repetitive activities for the lower layers of the Type 24 fieldbus, which of all the slave devices are synchronized with that of a C1 master device by the lower layer protocol

3.2.44
transmission mode
state or a way of transmission for the lower layer protocol of the Type 24 fieldbus; cyclic mode, event driven mode

3.2.45
virtual memory space
large data block of APOs for the Type 24 FAL that can be read and written with pseudo-memory-addresses to provide consistency between different device models

Note 1 to entry: The virtual memory space includes the device Information and other vender specific area, see Annex B.

3.2.46
warning
field device status to tell that the device has detected a slight or passing problem but is still working normally through the field device control (FDC) service of the Type 24 fieldbus

Note 1 to entry: Any warning statuses are latched and need to be operated to clear them.

Note 2 to entry: Warnings are classified into three groups, communication warnings, illegal-command-related ones, and application specific ones. But concrete definitions are dependent on an implementation of each field devices.

3.2.47
node
device that comply with the protocols described in this specification

3.2.48
output data
data to be sent from a master to a slave

3.2.49
input data
data to be sent from a slave to a master

3.2.50
node address
address to be specified as destination address or source address

3.3 Abbreviations and symbols

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

AE	Application Entity
AL	Application Layer
A-, AL-	Application Layer (as a prefix)
ALME	Application Layer Management Entity
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Invocation
APO	Application Process Object
APC	Application Process Context (as prefix of a protocol for Type 24 fieldbus)
APC SM	Application Process Context State Machine (for Type 24 fieldbus)
AR	Application Relationship
AR ASE	Application Relationship Application Service Element
AREP	Application Relationship End Point
ARPM	Application Relationship Protocol Machine (for Type 24 fieldbus)
ARPM-FDCM	ARPM for Field Device Control service Master (for Type 24 fieldbus)
ARPM-FDCMN	ARPM for Field Device Control service Monitor (for Type 24 fieldbus)
ARPM-FDCS	ARPM for Filed Device Control service Slave (for Type 24 fieldbus)
ARPM-MSG	ARPM for Message service (for Type 24 fieldbus)
ASCII	American Standard Code for Information Interchange
ASDU	Application Service Data Unit
ASE	Application Service Element
ASN.1	Abstract Syntax Notation One
CMD	Command PDU for FDC service (for Type 24 fieldbus)
Cnf	confirm primitive
DL-	(as a prefix) Data Link-
DLL	Data Link Layer
DLM	Data Link-management

DLPDU	Data Link-Protocol Data Unit
DLSAP	Data Link Service Access Point
DLSDU	Data Link Service Data Unit
DMPM	Data Link Mapping Protocol Machine
E2PROM	Electrically erasable programmable read only memory
FAL	Fieldbus Application Layer
FCS	Frame check sequence
FDC-	Field Device Control (as prefix of a service or a protocol for Type 24 fieldbus)
FDC ASE	Field Device Control Application Service Element (for Type 24 fieldbus)
FDCPM	Field Device Control Protocol Machine (for Type 24 fieldbus)
FDCPM-M	Field Device Control Protocol Machine for Master (for Type 24 fieldbus)
FDCPM-MN	Field Device Control Protocol Machine for Monitor (for Type 24 fieldbus)
FDCPM-S	Field Device Control Protocol Machine for Slave (for Type 24 fieldbus)
FIFO	First In First Out
FSPM	FAL service protocol machine
FSM-	Fieldbus System Management (as prefix of a service for Type 24 fieldbus)
FSM ASE	Fieldbus System Management Application Service Element for Type 24 fieldbus
HMI	Human-machine Interface
I/O	Input/output
ID	Identifier
Ind	indication primitive
LME	Layer Management Entity
Lsb	least significant bit
MAC	Media Access Control
Msb	most significant bit
MSG	Message (as prefix of a service or a protocol for Type 24 fieldbus)
MSG ASE	Message Application Service Element for Type 24 fieldbus
MSGPM	Message Protocol Machine for Type 24 fieldbus
MSGPM-RQ	MSGPM for Requester for Type 24 fieldbus
MSGPM-RS	MSGPM for Responder for Type 24 fieldbus
OSI	Open Systems Interconnection
PM	Protocol machine
PDU	Protocol Data Unit
PhL	Ph-layer
QoS	Quality of Service
RAM	Random access memory
Req	request primitive
Rsp	response primitive
RSP	Response PDU for FDC service (for Type 24 fieldbus)
SAP	Service Access Point
SDN	Send Data with no Acknowledge
SDU	Service Data Unit

SM	State Machine
SMIB	System Management Information Base
UML	Unified Modelling Language

3.4 Conventions

3.4.1 General conventions

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158-5-24. The protocol specification for each of the ASEs is defined in this document.

The class definitions define the attributes of the classes supported by each ASE. In this document, it is assumed that all attributes are accessible only from the owner's instance object itself directly or through services of the class.

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

3.4.2 PDU data type conventions

The data types of FAL PDUs are defined with the notations of ASN.1.

Character strings that begin with "_" (low line) are used as data type symbols of FAL PDUs or their fields. A same string as a PDU type without "_" means a PDU instance of the corresponding data type.

Example CMD-PDU means an instance of _CMD-PDU and a following statement is omitted.

```
CMD-PDU  _CMD-PDU ::= { value }
```

3.4.3 State machine conventions

The protocol sequences are described by means of State Machines.

In statechart diagrams, states are represented as boxes, and state transitions are shown as arrows. Their conventions are given in ISO/IEC 19501 as Universal Modeling Language (UML).

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 Table 1.

The first row contains the name of the transition by an index number.

The second row defines the source state or the current state before the transition.

The third row contains an event and actions. The event is followed by optional arguments in parentheses, "(" and ")", and guard conditions in brackets, "[" and "]", as the first line. And the actions with starting character "/" follow the event line.

The last row contains the target state or the next state after the transition.

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

Table 1 – State transition descriptions

T#	Source State	Event (arguments) [conditions] / Action	Target state

A meaning of each element in Table 1 is shown in Table 2, based on UML "transition" definition (ISO/IEC 19501).

Table 2 – Description of state machine elements

Description element	Meaning
Source state Target state	Names of the originating state and the target state of transition.
T#	Name or number of the transition.
Event	Name or description of the trigger event that fire the transition.
(arguments)	A parameter value, an expression, or a sequence of those divided by "," for the event. The preceding "(" and the succeeding ")" are not part of the argument list.
[conditions]	Boolean expression, which is true for the transition to be fired. The preceding "[" and the succeeding "]" are not part of the condition.
/ Action	List of assignments and service or function invocations. The action should be atomic. The preceding "/" is not part of the action.
NOTE "(arguments)" and "[condition]" can be omitted if not necessary.	

The conventions used in the state machines are shown in Table 3.

Table 3 – Conventions used in state machines

Convention	Meaning
/*-- description --*/	Describes and explains conditions and/or procedure by using normal sentence between "/*--" and "--*/" instead of using the pseudo code notation. Example: /*-- The PM examines an occurrence of any alarms for the corresponding remote device. -- If an alarm is detected, the PM notifies it to the FSM ASE. -- If not, the PM transfers the MSGService-PDU to the DLL. --*/
=	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	Parameter name if ‘a' is a letter Example: Identifier = reason; means value of a 'reason' parameter is assigned to a parameter called 'Identifier'
"xxx"	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.
==	Logical condition to indicate an item on the left is equal to an item on the right
<	Logical condition to indicate an item on the left is less than the item on the right
>	Logical condition to indicate an item on the left is greater than the item on the right
!=	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

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

4 Abstract syntax

4.1 Basic Data types

The following data types are used in the Type 24 FAL.

- Basic types as simple types of ASN.1:
  - INTEGER;
  - REAL;
  - BIT STRING;
  - OCTET STRING;
  - NULL.
- Specific basic types as subtypes from ASN.1:
  - Integer8::= INTEGER (−128..127);
  - Integer16::= INTEGER (−32 768 .. 32 767);
  - Integer32::= INTEGER (−2 147 483 648 .. 2 147 483 647);
  - Integer64::= INTEGER ((−1)× ‘8000 0000 0000 0000'H..‘7FFF FFFF FFFF FFFF'H);
  - Unsigned8 ::= INTEGER (0..'FF'H);
  - Unsigned16::= INTEGER (0..'FFFF'H);
  - Unsigned32::= INTEGER (0..'FFFF FFFF'H);
  - Unsigned64::= INTEGER (0..'FFFF FFFF FFFF FFFF'H);
  - $Float_{32}$::= 0 | negative infinity | positive infinity

    | REAL (WITH COMPONENTS { mantissa ((−1) × c_{32}), base (2), exponent (q_{32}) })

    | REAL (WITH COMPONENTS { mantissa (c_{32}), base (2), exponent (q_{32}) })

    where

    c_{32}::= REAL (1.. $\left(\sum_{i=0}^{23} \frac{1}{2^i}\right)$) = REAL (1..($2-2^{-23}$)), and

    q_{32}::= INTEGER (−127..127).

NOTE 1 The range of the real value is covered from negative infinity to positive infinity. But the representable precision conforms to binary32 or the single precision of ISO/IEC/IEEE 60559, see 5.2.2.

  - $Float_{64}$::= 0 | negative infinity | positive infinity

    | REAL (WITH COMPONENTS { mantissa ((−1) × c_{64}), base (2), exponent (q_{64}) })

    | REAL (WITH COMPONENTS { mantissa (c_{64}), base (2), exponent (q_{64}) })

    where

    c_{64}::= REAL (1..$\left(\sum_{i=0}^{52} \frac{1}{2^i}\right)$) = REAL (1.. ($2-2^{-52}$)), and

    q_{64}::= INTEGER (−1 023..1 023).

NOTE 2 The range of the real value is covered from negative infinity to positive infinity. But the representable precision conforms to binary64 or the double precision of ISO/IEC/IEEE 60559, see 5.2.2.

  - IA5String
- Structure types with component types of ASN.1:

  Structure type is defined by a combination of SEQUENCE type, CHOICE type of ASN.1, basic types and subtype elements defined in ASN.1. The FAL PDU described in 4.2 and 4.3 that follows this subclause correspond this Structure type.
  - SEQUENCE
  - CHOICE

- Array types with a component type of ASN.1:

  Array types are a list of any data type defined with SEQUENCE OF type of ASN.1. Typical array types are shown with SEQUENCE OF type, below:

  – SEQUENCE OF;

  – BitArray::= SEQUENCE OF (BIT STRING SIZE(8));

  – Array8::= SEQUENCE OF Integer8;

  – Array16::= SEQUENCE OF Integer16;

  – Array32::= SEQUENCE OF Integer32;

  – Array64::= SEQUENCE OF Integer64.

4.2 FAL PDU types

4.2.1 Top of APDU types: _APDU

The APDU type: _APDU shall consist of two groups; _FDCServicePDU and _MSGServicePDU.

And _FDCServicePDU shall consist of a pair of PDUs; _CMD-PDU and _RSP-PDU, used for FDC commands and its responses respectively.

Furthermore, _MSGServicePDU shall consist of a pair of PDUs; _MSGREQ-PDU and _MSGRSP-PDU, used for MSG requests and its responses respectively.

```
 _APDU                      ::= _FDCServicePDU
                               | _MSGServicePDU
 _FDCServicePDU             ::= _CMD-PDU              -- FDC command PDU type
                               | _RSP-PDU             -- FDC response PDU type
 _MSGServicePDU             ::= _MSGREQ-PDU           -- MSG request PDU type
                               | _MSGRSP-PDU          -- MSG response PDU type
```

Figure 1 shows the overview of APDU type definitions by the tree structure chart.

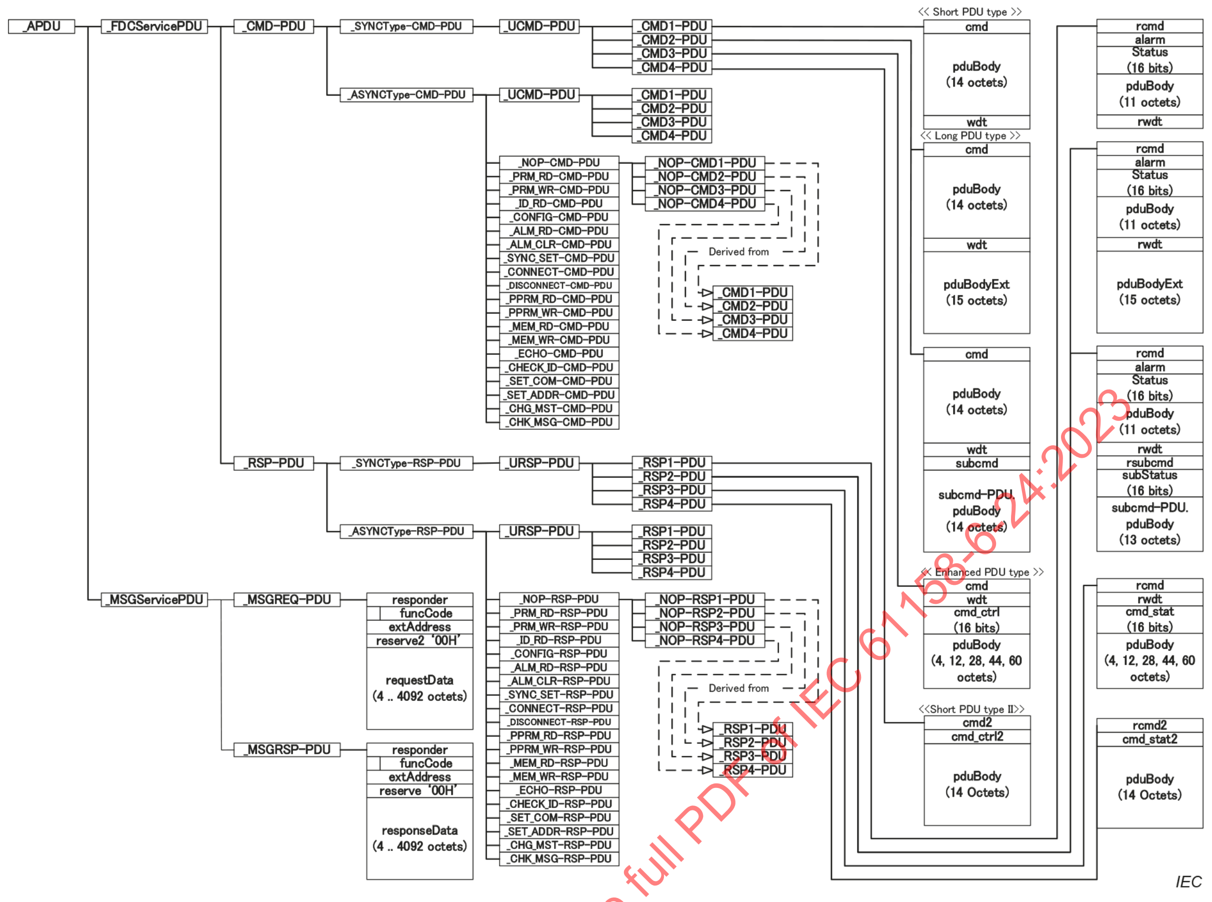


Figure 1 – Tree structure of APDU types

4.2.2 PDUs for field device control service

4.2.2.1 Overview

FDCServicePDU shall be edited with a SDU in the FDC service primitive and be sent via AR. And a FDC ASE receives the PDU from the peer ASE via AR.

The FDC command PDU type: _CMD_PDU shall be classified into sync type command (sync command): _SYNCType-CMD-PDU and async type command (async command): _ASYNCType-CMD-PDU. Both types are the identical PDU formats: _UCMD-PDU. Therefore, the FDC PM cannot distinguish their type's PDUs. The user of FDC ASE should know which commands belong to the sync commands or not, and can distinguish the types by using cmd field as a key code in the PDU header part.

The async command PDU: _ASYNCType-CMD_PDU shall include _UCMD-PDU and other twenty PDU types as common commands, which are also variants of _UCMD-PDU.

Derived from the _UCMD-PDU and the _URSP-PDU explained later, a set of commands and responses can be defined for a specific application, such as motor-drive control, servo control, etc., by FAL users, and the set can be registered as a device profile (see 4.4 and Annex A). In addition, a FAL user can select any device profiles supported by a target field device, when a connection is established or when it is transited to the cyclic communication mode for connection-less type by FDC-Connect service (see 8.2.4.3 and IEC 61158-5-24, 6.4.1.2.3.4).

```
_CMD-PDU                    ::= _SYNCType-CMD-PDU
                              | _ASYNCType-CMD-PDU
_SYNCType-CMD-PDU           ::= _UCMD-PDU
_ASYNCType-CMD-PDU          ::= _UCMD-PDU
                              | _NOP-CMD-PDU
                              | _PRM_RD-CMD-PDU
                              | _PRM_WR-CMD-PDU
                              | _ID_RD-CMD-PDU
                              | _CONFIG-CMD-PDU
                              | _ALM_RD-CMD-PDU
                              | _ALM_CLR-CMD-PDU
                              | _SYNC_SET-CMD-PDU
                              | _CONNECT-CMD-PDU
                              | _DISCONNECT-CMD-PDU
                              | _PPRM_RD-CMD-PDU
                              | _PPRM_WR-CMD-PDU
                              | _MEM_RD-CMD-PDU
                              | _MEM_WR-CMD-PDU
                              | _ECHO-CMD-PDU
                              | _CHECK_ID-CMD-PDU
                              | _SET_COM-CMD-PDU
                              | _SET_ADDR-CMD-PDU
                              | _CHG_MST-CMD-PDU
                              | _CHK_MSG-CMD-PDU
_UCMD-PDU                   ::= _CMD1-PDU | _CMD2-PDU | _CMD3-PDU | _CMD4-PDU
_NOP-CMD-PDU                ::= _NOP-CMD1-PDU
                              | _NOP-CMD2-PDU
                              | _NOP-CMD3-PDU
                              | _NOP-CMD4-PDU
_PRM_RD-CMD-PDU             ::= _PRM_RD-CMD1-PDU | _PRM-RD-CMD2-PDU | _PRM_RD-CMD3-PDU
_PRM_WR-CMD-PDU             ::= _PRM_WR-CMD1-PDU | _PRM_WR-CMD2-PDU | _PRM_WR-CMD3-PDU
_ID_RD-CMD-PDU              ::= _ID_RD-CMD1-PDU | _ID_RD-CMD2-PDU | _ID_RD-CMD3-PDU
_CONFIG-CMD-PDU             ::= _CONFIG-CMD1-PDU | _CONFIG-CMD2-PDU | _CONFIG-CMD3-PDU
_ALM_RD-CMD-PDU             ::= _ALM_RD-CMD1-PDU | _ALM_RD-CMD2-PDU | _ALM_RD-CMD3-PDU
_ALM_CLR-CMD-PDU            ::= _ALM_CLR-CMD1-PDU | _ALM_CLR-CMD2-PDU | _ALM_CLR-CMD3-PDU
_SYNC_SET-CMD-PDU           ::= _SYNC_SET-CMD1-PDU
                              | _SYNC_SET-CMD2-PDU
                              | _SYNC_SET-CMD3-PDU
_CONNECT-CMD-PDU            ::= _CONNECT-CMD1-PDU
                              | _CONNECT-CMD2-PDU
                              | _CONNECT-CMD3-PDU
_DISCONNECT-CMD-PDU         ::= _DISCONNECT-CMD1-PDU
                              | _DISCONNECT-CMD2-PDU
                              | _DISCONNECT-CMD3-PDU
_PPRM_RD-CMD-PDU            ::= _PPRM_RD-CMD1-PDU | _PPRM-RD-CMD2-PDU | _PPRM_RD-CMD3-PDU
_PPRM_WR-CMD-PDU            ::= _PPRM_WR-CMD1-PDU | _PPRM_WR-CMD2-PDU | _PPRM_WR-CMD3-
                            PDU
```

```
_MEM_RD-CMD-PDU           ::= _MEM_RD-CMD3-PDU
_MEM_WR-CMD-PDU           ::= _MEM_WR-CMD3-PDU
_ECHO-CMD-PDU             ::= _ECHO_ID-CMD4-PDU
_CHK_ID-CMD-PDU           ::= _CHK_ID-CMD4-PDU
_SET_COM-CMD-PDU          ::= _SET_COM-CMD4-PDU
_SET_ADDR_ID-CMD-PDU      ::= _SET_ADDR-CMD4-PDU
_CHG_MST-CMD-PDU          ::= _CHG_MST-CMD4-PDU
```

The FDC response PDU type: _RSP_PDU shall be also classified into sync type response (sync response): _SYNCType-RSP-PDU and async type response (async response): _ASYNCType-RSP-PDU. Both types shall be derived from the identical PDU formats: _URSP-PDU. Therefore, the FDC PM cannot distinguish their type's PDUs. The user of FDC ASE should know which responses belong to the sync responses or not, and can distinguish the types by using rcmd field as a key code in the PDU header part. Moreover, this prefix code of a response PDU corresponds to one of a command PDU, since a response shall be exchanged for a command.

The async type response PDU: _ASYNCType-RSP_PDU shall include _URSP-PDU and other twenty PDU types as common responses to any application, which are also variants of _URSP-PDU.

```
_RSP-PDU                  ::= _SYNCType-RSP-PDU
                            | _ASYNCType-RSP-PDU
_SYNCType-RSP-PDU         ::= _URSP-PDU
_ASYNCType-RSP-PDU        ::=_URSP-PDU
                            | _NOP-RSP-PDU
                            | _PRM_RD-RSP-PDU
                            | _PRM_WR-RSP-PDU
                            | _ID_RD-RSP-PDU
                            | _CONFIG-RSP-PDU
                            | _ALM_RD-RSP-PDU
                            | _ALM_CLR-RSP-PDU
                            | _SYNC_SET-RSP-PDU
                            | _CONNECT-RSP-PDU
                            | _DISCONNECT-RSP-PDU
                            | _PPRM_RD-RSP-PDU
                            | _PPRM_WR-RSP-PDU
                            | _MEM_RD-RSP-PDU
                            | _MEM_WR-RSP-PDU
                            | _ECHO-RSP-PDU
                            | _CHECK_ID- RSP-PDU
                            | _SET_COM- RSP-PDU
                            | _SET_ADDR- RSP-PDU
                            | _CHG_MST- RSP-PDU
                            | _CHK_MSG- RSP -PDU
_URSP-PDU                 ::= _RSP1-PDU | _RSP2-PDU | _RSP3-PDU  | _RSP4-PDU
_NOP-RSP-PDU              ::= _NOP-RSP1-PDU
                            | _NOP-RSP2-PDU
                            | _NOP-RSP3-PDU
                            | _NOP-RSP4-PDU
```

```
_PRM_RD-RSP-PDU          ::= _PRM_RD-RSP1-PDU | _PRM-RD-RSP2-PDU | _PRM_RD-RSP3-PDU
_PRM_WR-RSP-PDU          ::= _PRM_WR-RSP1-PDU | _PRM_WR-RSP2-PDU | _PRM_WR-RSP3-PDU
_ID_RD-RSP-PDU           ::= _ID_RD-RSP1-PDU | _ID_RD-RSP2-PDU | _ID_RD-RSP3-PDU
_CONFIG-RSP-PDU          ::= _CONFIG-RSP1-PDU | _CONFIG-RSP2-PDU | _CONFIG-RSP3-PDU
_ALM_RD-RSP-PDU          ::= _ALM_RD-RSP1-PDU | _ALM_RD-RSP2-PDU | _ALM_RD-RSP3-PDU
_ALM_CLR-RSP-PDU         ::= _ALM_CLR-RSP1-PDU | _ALM_CLR-RSP2-PDU | _ALM_CLR-RSP3-PDU
_SYNC_SET-RSP-PDU        ::= _SYNC_SET-RSP1-PDU
                           | _SYNC_SET-RSP2-PDU
                           | _SYNC_SET-RSP3-PDU
_CONNECT-RSP-PDU         ::= _CONNECT-RSP1-PDU
                           | _CONNECT-RSP2-PDU
                           | _CONNECT-RSP3-PDU
_DISCONNECT-RSP-PDU      ::= _DISCONNECT-RSP1-PDU
                           | _DISCONNECT-RSP2-PDU
                           | _DISCONNECT-RSP3-PDU
_PPRM_RD-RSP-PDU         ::= _PPRM_RD-RSP1-PDU | _PPRM-RD-RSP2-PDU | _PPRM_RD-RSP3-PDU
_PPRM_WR-RSP-PDU         ::= _PPRM_WR-RSP1-PDU | _PPRM_WR-RSP2-PDU | _PPRM_WR-RSP3-PDU
_MEM_RD-RSP-PDU          ::= _MEM_RD-RSP3-PDU
_MEM_WR-RSP-PDU          ::= _MEM_WR-RSP3-PDU
_ECHO-RSP-PDU            ::= _ECHO-RSP4-PDU
_CHK_ID-RSP-PDU          ::= _CHK_ID-RSP4-PDU
_SET_COM- RSP-PDU        ::= _SET_COM-RSP4-PDU
_SET_ADDR- RSP-PDU       ::= _SET_ADDR-RSP4-PDU
_CHG_MST- RSP-PDU        ::= _CHG_MST-RSP4-PDU
_MSG_REQ- RSP-PDU        ::= _MSG_REQ-RSP4-PDU
```

4.2.2.2 Common component of PDU

4.2.2.2.1 FDC command PDU

_UCMD-PDU has four types, short type: _CMD1-PDU, long type: _CMD2-PDU, and enhanced type: _CMD3-PDU, and short type II: _CMD4-PDU. Both _CMD1-PDU and _CMD2-PDU shall have same header fields, but _CMD3-PDU shall have another enhanced one.

The pduBody length of _CMD1-PDU shall be 14 octets when it encoded, and so as for _CMD2-PDU. This part is called main-command, and _CMD2-PDU shall have another extended pduBodyExt field too, and can piggyback an additional command PDU called sub-command: _SUBCMD-PDU on that field.

The pduBody length of_CMD3-PDU shall be selected from 4, 12, 28, 44, and 60 octets. It can be selected for each station; however, it shall not be changed after selecting.

The pduBody length of _CMD4-PDU shall be within the range of 1 octet to 14 octets. It can be changed for each _CMD4-PDU.

```
_CMD1-PDU          ::= SEQUENCE{                                          -- short PDU type
                         cmd        _CMDCode,
                         pduBody  OCTET STRING SIZE (14) ,
                         wdt        _WDT }
_CMD2-PDU          ::= SEQUENCE{                                          -- long PDU type
                         maincmd-PDU   COMPONENTS OF _CMD1-PDU,           -- main-command
                         CHOICE { subcmd-PDU _SUBCMD-PDU,                  -- sub-command
                                  pduBodyExt   OCTET STRING SIZE (15) } }
_SUBCMD-PDU        ::= _USUBCMD-PDU  SEQUENCE{                            -- sub command
                                  subcmd    _SubCMD,                       PDU type
                                  pduBody  OCTET STRING SIZE
                   (14)}
                     | _NOP-SUBCMD-PDU
                     | _PRM_RD-SUBCMD-PDU
                     | _PRM_WR-SUBCMD-PDU
                     | _ALM_RD-SUBCMD-PDU
                     | _PPRM_RD-SUBCMD-PDU
                     | _PPRM_WR-SUBCMD-PDU
_CMD3-PDU          ::= SEQUENCE{                                          -- enhanced PDU
                         cmd        _CMDCode,                              type
                         wdt        _WDT,
                         cmd_ctrl   _CMD_CTRL,
                         pduBody   OCTET STRING SIZE (4|12|28|44|60) }
_CMD4-PDU          ::= SEQUENCE{                                          -- short PDU type
                         cmd2          _CMDCode2,                          ||
                         cmd_ctrl2  _CMD_CTRL2,
                         pduBody     OCTET STRING SIZE (1..14) }
```

- cmd field

cmd
data field which contains the code to identify contents of each command

This field shall be coded as _CMDCode, subtype of Unsigned8, with the following value range. The predefined common commands are shown as _PrimaryCMD below.

– '00'H to '1F'H: used or reserved for common commands;
– '20'H to 'BF'H: reserved for application specific commands;
– 'C0'H to 'FF'H: reserved for vendors.

```
_CMDCode                    ::= Unsigned8
_PrimaryCMD                 ::= _CMDCode {
                                    nop ('00'H),
                                    prm_rd ('01'H), prm_wr ('02'H),
                                    id_rd ('03'H),
                                    config ('04'H),
                                    alm_rd ('05'H), alm_clr ('06'H),
                                    sync_set ('0D'H),
                                    connect ('0E'H),
                                    disconnect ('0F'H),
                                    pprm_rd ('1B'H), pprm_wr ('1C'H),
                                    mem_rd ('1D'H), mem_wr ('1E'H) }
```

- wdt field

wdt
watchdog counter field

This field shall be used for the slave to detect out-of synchronous activity or the WDT error of the corresponding master. The master shall count the wdt field of sending PDUs every communication cycle in the case of the synchronous communication state. In addition, the slave shall examine the field value of receiving PDUs every communication cycle to detect the wdt error.

The Data type is _WDT. It shall have two sub fields, mn and sn.

```
_WDT                        ::= SEQUENCE { mn  BIT STRING SIZE (4),
                                           sn  BIT STRING  SIZE (4)}
```

mn (Master count value)
The value range shall be from 0 to 15.

The mn field in a next CMD-PDU to be sent shall be counted up by one. An FDC master shall do that every communication cycle. On the other hand, when an FDC slave receives the PDU, it shall examine the counter every cycle. And, if the counter isn't matching the last mn value plus one, it shall alert the watchdog counter error.

sn (Slave count value)
The value range shall be from 0 to 15.

The sn field in a next CMD-PDU to be sent shall be set the same value of the rsn field in the last received RSP-PDU. An FDC master shall copy it from the last RSP-PDU every communication cycle.

- subcmd field

subcmd
data field which contains the code to identify contents of each sub-command

The Data type is _SubCMD. The code definition for the sub-commands shall be the same as main cmd field in principle, but some codes are restricted to use, if they have any difficulties in processing parallel commands. The predefined common sub-commands are shown as _PrimarySubCMD below.

```
_SubCMD               ::= Unsigned8
_PrimarySubCMD        ::= _SubCMD {
                              nop (‘00'H),
                              prm_rd (‘01'H), prm_wr (‘02'H),
                              alm_rd (‘05'H),
                              pprm_rd (‘1B'H), pprm_wr (‘1C'H) }
```

- cmd_ctrl field

cmd_ctrl
set of control bits that is independent of any command transactions, from FDC master to FDC slave

The Data type is _CMD_CTRL. It shall have two meaningful sub-fields, alm_clr and cmd_id.

```
_CMD_CTRL             ::= SEQUENCE { reserve1 BIT STRING SIZE (3),
                                     alm_clr   BIT STRING SIZE (1),
                                     reserve2 BIT STRING SIZE (2),
                                     cmd_id    BIT STRING SIZE (2),
                                     reserve3 BIT STRING SIZE (8) }
```

alm_clr
This field shall be used to command to clear alarm/warning status of the slave device.

value 0: to execute nothing

value changes to1: to execute the status clear

cmd_id
When the master device continuously issues identical contents of a CMD-PDU to the slave, it can be used to make the slave device recognizing the command as another command.

The value range shall be from 0 to 3.

The FDC master can use the cmd_id to have a FDC slave acknowledge that the command is a new command when the master sends the same command repeatedly to the slave.

Since the slave shall respond the echo of the cmd_id of every command through RSP-PDU.cmd_stat.rcmd_id field explained later, the master can also judge the command for which the slave station sent the response.

Use of the cmd_id is not mandatory for the master.

While RSP-PDU.cmd_stat.cmdRdy = 0 in response PDUs, the slave shall disregard commands that have a different cmd_id and continues to execute the command which has been already accepted.

- cmd2 field

cmd2
data field which contains the code to identify contents of each command

This field shall be coded as _CMDCode2, subtype of Unsigned8, with the following value range. The predefined common commands are shown as _PrimaryCMD2 below.

```
_CMDCode2                ::= Unsigned8
_PrimaryCMD2             ::= _CMDCode2 {
                             nop (‘00'H),
                             echo (‘FF'H),
                             check_id (‘E2'H),
                             set_com (‘FA'H),
                             set_addr (‘F3'H),
                             chg_mst (‘0E'H),
                             chk_msg_req (‘0F'H) }
```

- cmd_ctrl2 field

cmd_ctrl2
set of control data that is independent of any command transactions, from FDC master to FDC slave. The Data type is _CMD_CTRL2 and cmd_ctrl2 depends on cmd2.

```
_CMD_CTRL2               ::= Unsigned8
```

4.2.2.2.2 FDC response PDU

_URSP-PDU has four types, short type: _RSP1-PDU, long type: _RSP2-PDU, and enhanced type: _RSP3-PDU, and short type II: _RSP4-PDU. Both _RSP1-PDU and _RSP2-PDU shall have common header fields, but _RSP3-PDU shall have another enhanced one.

The pduBody length of _RSP-PDU shall be 11 octets, and so as for _RSP2-PDU. This is called a main response or a response of the main command, and _RSP2-PDU shall have another extended pduBodyExt field too, and can piggyback an additional command PDU called sub-response or a response of a sub command: _SUBRS-PDU on that field.

The pduBody length of_RSP3-PDU shall be able to be selected from 4, 12, 28, 44, and 60 octets It can be selected for each station; however, it shall not be changed after selecting.

The pduBody length of _RSP4-PDU shall be within the range of 1 octet to 14 octets. It can be changed for each _RSP4-PDU.

```
_RSP1-PDU               ::= SEQUENCE{                                   -- short PDU type
                            rcmd        _CMDCode,
                            alarm       _ALARM,
                            status      _STATUS,
                            pduBody     OCTET STRING SIZE (11),
                            rwdt        _RWDT}
_RSP2-PDU               ::= SEQUENCE{                                   -- long PDU type
                              rsp1-PDU COMPONENTS OF _RSP1-PDU,         -- main response
                              CHOICE { subrsp-PDU _SUBRS-PDU,           -- sub response
                                       pduBodyExt  OCTET STRING SIZE (15) } }
```

```
_SUBRSP-PDU          ::= _USUBRSP-PDU  SEQUENCE{                    -- sub response
                                  rsubcmd   _SubCMD,                  PDU type
                                  subStatus  _SUBSTATUS,
                                  pduBody    OCTET STRING SIZE
                     (13)}
                      | _NOP-SUBRSP-PDU
                      | _PRM_RD-SUBRSP-PDU
                      | _PRM_WR-SUBRSP-PDU
                      | _ALM_RD-SUBRSP-PDU
                      | _PPRM_RD-SUBRSP-PDU
                      | _PPRM_WR-SUBRSP-PDU
_RSP3-PDU            ::= SEQUENCE{                                   -- enhanced PDU
                         rcmd       _CMDCode,                          type
                         rwdt       _RWDT,
                         cmd_stat   _CMD_STAT,
                         pduBody    OCTET STRING SIZE (4|12|28|44|60) }
_RSP4-PDU            ::= SEQUENCE{                                   -- short PDU type
                         rcmd2      _CMDCode2,                         II
                         cmd_stat2  _CMD_STAT2,
                         pduBody    OCTET STRING SIZE (1..14) }
```

- rcmd field

rcmd
data field which contains the same code as a received and processing command

So, this field shall be coded as _CMDCode, see 4.2.2.2.1.

- alarm field

alarm
This field shall contain the alarm code.

The Data type is _ALARM. The value range shall be from 0 to 255.

```
_ALARM               ::= Unsigned8
```

- status field

status
This field shall indicate the status of the slave device.

The Data type is _STATUS. It shall have three meaningful sub-fields as follows.

```
_STATUS              ::= SEQUENCE { alarm    BIT STRING SIZE (1),
                                    warning  BIT STRING SIZE (1),
                                    cmdRdy   BIT STRING SIZE (1),
                                    reserve  BIT STRING SIZE  (13) }
```

alarm
This bit field shall indicate the alarm status in the slave device.

0: No alarm,

1: any alarms occur.

warning
This bit field shall indicate the warning status in the slave device.

0: No warning,

1: any warnings occur.

cmdRdy
This bit field shall indicate the command progress status of the FDC slave.

The slave device shall keep the bit cmdRdy 0, while it is processing the command. The slave device shall change the bit from 0 to 1, when it has completed the processing.

To notice the completion of the specific command, the FDC master shall not only examine the bit, but also collate some other RSP-PDU fields with the corresponding CMD-PDU fields to discriminate from the preceding transaction.

When the retention time of cmdRdy = 0 is expired, the master shall raise a command time out error. The timer value depends on each product specification for slave devices.

A change of this bit status shall be independent of the alarm or warning status.

0: busy with command execution in progress

1: ready for new command

- rwdt field

rwdt
watchdog counter field

This field shall be used for the master to detect out-of synchronous activity or the WDT error of the corresponding slave. The slave shall count and update the rwdt field of sending PDUs every communication cycle in the case of the synchronous communication state. In addition, the master shall examine the field value of receiving PDUs every communication cycle to detect the rwdt error.

The Data type is _RWDT. It shall have two sub fields, rmn and rsn.

```
_RWDT                    ::= SEQUENCE { rmn  BIT STRING SIZE (4),
                                        rsn  BIT STRING  SIZE (4)}
```

rmn (Master count value)
The value range shall be from 0 to 15.

The rmn field in a next RSP-PDU to be sent shall be set the same value of the mn field in the last received CMD-PDU. An FDC slave shall copy it from the CMD-PDU every communication cycle.

rsn (Slave count value)
The value range shall be from 0 to 15.

The rsn field in a next RSP-PDU to be sent shall be counted up by one. An FDC slave shall do that every communication cycle. On the other hand, when an FDC master receives the PDU, it shall examine the counter every cycle. And, if the counter isn't matching the last rsn value plus one, it shall alert the watchdog counter error.

- rsubcmd

rsubcmd
data field which contains the same code as the received and processing sub-command

This field shall be coded as _SubCMD, see 4.2.2.2.1.

- subStatus

subStatus
This field shall indicate the status of the slave station in the sub response PDU part.

The Data type is _SubSTATUS. This field shall be coded same as status field.

```
_SUBSTATUS                 ::= _STATUS
```

- cmd_stat field

cmd_stat
This field shall indicate the status of the slave device.

The Data type is _CMD_STAT. It shall have seven sub fields as follows.

```
_CMD_STAT                  ::= SEQUENCE { d_alm     BIT STRING SIZE (1),
                                          d_war     BIT STRING SIZE (1),
                                          cmdRdy BIT STRING SIZE (1),
                                          alm_clr_cmp BIT STRING SIZE (1),
                                          reserve  BIT STRING SIZE (2),
                                           rcmd_id  BIT STRING SIZE (2),
                                           cmd_alm BIT STRING SIZE (4),
                                          comm_alm BIT STRING SIZE (4) }
```

d_alm
This bit field shall indicate occurrence status of device-specific alarm in the slave device.

0: the slave is not in device specific alarm status;

1: the slave is in device specific alarm status except comm_alm or cmd_alm.

Specification of device alarm status depends on specified product implementation, but a factor of the alarms should be classified separately from a communication problem and a command issue.

After the slave has been recovered to the alarm status, through the execution of the command ALM_CLR-CMD-PDU or cmd_ctrl.alm_clr bit in any CMD-PDU, this bit shall be cleared to 0.(d_alm = 0)

d_war
This bit field shall indicate occurrence status of device-specific warning in the slave station.

0: the slave is not in device specific alarm status;

1: the slave is in device specific alarm status except comm_alm or cmd_alm.

Specification of device alarm status depends on specified product implementation, but a factor of the alarms should be classified separately from a communication problem and a command issue.

After the slave has been recovered to the warning status, through the execution of the command ALM_CLR-CMD-PDU or cmd_ctrl.alm_clr bit in any CMD-PDU, this bit shall be cleared to 0.(d_war = 0)

cmdRdy
This bit field shall indicate the command progress status of the FDC slave.

Meaning of this bit is the same as RSP1_PDU.status.cmdRdy.

0: busy with command execution in progress;

1: ready for new command.

alm_clr_cmp
This bit field shall indicate the execution status of alarm clear process.

0: Alarm/warning clear not completed;

1: Alarm/warning clear completed.

rcmd_id
This field shall be a key code to indicate which command the response PDU corresponds to, by echoing back cmd_id of the corresponding command PDU.

The value range is from 0 to 3.

cmd_alm
This field shall notify an alarm code for command abnormality.

Each code shall mean the following.

'00'H: Normal;

'01'H: Warning on out-of-range data;

'02'H to '07'H: reserved for warning codes;

'08'H: Alarm on out-of-support;

'09'H: Alarm on out-of-range data;

'0A'H: Alarm on abnormal command execution condition;

'0B'H: Alarm on abnormal subcommand combination;

'0C'H: Alarm on abnormal phase;

'0D'H to '0F'H: reserved for alarm codes.

comm_alm
This field shall notify an alarm code for communication error status.

Each code shall mean the following.

'00'H: Normal;

'01'H: Warning on abnormal FCS;

'02'H: Warning on abnormal reception;

'03'H to '07'H: reserved for warning;

'08'H: Alarm on abnormal FCS;

'09'H: Alarm on abnormal reception;

'0A'H to '0F'H: reserved for alarm.

- rcmd2 field

rcmd2
data field which contains the same code as a received and processing command.

So, this field shall be coded as _CMDCode2, see 4.2.2.2.1.

- cmd_stat2 field

cmd_stat2
This field shall indicate the status of the slave device.

The Data type is _CMD_STAT2. It shall have seven sub fields as follows.

```
_CMD_STAT2          ::= SEQUENCE { busy BIT STRING SIZE (1),
                                   com_err BIT STRING SIZE (1),
                                   reserve BIT STRING SIZE (4),
                                   wng_flg BIT STRING SIZE (1),
                                   alm_flg BIT STRING SIZE (1) }
```

busy
This bit field shall indicate the command progress status of the FDC slave.

Meaning of this bit is the same as RSP1_PDU.status.cmdRdy.

0: busy with command execution in progress;

1: ready for new command.

comm_err
This bit field shall indicate the occurrence of command processing error or receiving of the unsupported command in the slave station.

wng_flg
This bit field shall indicate occurrence status of device-specific warning in the slave station.

0: the slave is not in device specific warning status;

1: the slave is in device specific warning status.

alm_flg
This bit field shall indicate occurrence status of device-specific alarm in the slave station.

0: the slave is not in device specific alarm status;

1: the slave is in device specific alarm status.

4.2.3 PDUs for message service

```
_MSGREQ-PDU          ::= SEQUENCE { responder  Unsigned8,
                                    funcCode BIT STRING SIZE (7),
                                    reserve1    BIT STRING SIZE (1),
                                    extAddress Unsigned8,
                                    reserve2    OCTET STRING (‘00'H),
                                    requestData OCTET STRING SIZE (4..4092)}
                       | SEQUENCE { requestUData OCTET STRING SIZE (8..4096)}
_MSGRSP-PDU          ::= SEQUENCE { responder  Unsigned8,
                                    funcCode BIT STRING SIZE (7),
                                    errorFlag  BIT STRING SIZE (1),
                                    extAddress Unsigned8,
                                    reserve      OCTET STRING (‘00'H),
                                  responseData OCTET STRING SIZE (4..4092)}
                       | SEQUENCE { responseUData OCTET STRING SIZE (8..4096)}
```

- responder field

responder
This field shall contain the destination node address or responder of the message.

This field shall be coded as data type Unsigned8 with the following value range.

- '00'H: reserved;
- '01'H: C1 master device;
- '02'H: C2 master device;
- '03'H to 'EF'h: Slave device;
- 'F0'H to 'FF'H: reserved.

- funcCode field

funcCode
This field shall contain the code indicating the function of the message.

The code range shall be from '00'H to '7F'H.

- errorFlag field

errorFlag
This field shall indicate the response state of the message. It shall be set to 0 on a normal response, and to 1 on an abnormal response.

  – 0: a normal response;
  – 1: an error response.

- extAddress field

extAddress
This field shall contain the sub address representing a sub device when the destination node is an integrated device with two or more sub devices.

This field shall be coded as data type Unsigned8 with the value range '00'H to 'FE'H. The code 'FF'H is reserved.

- requestData field and responseData field

requestData and responseData
These fields shall contain the SDUs that is defined by user for each function code. An example of a user message command set for function code: '42'H is shown in Annex C.

4.3 Detailed definitions of _FDCService-PDUs

4.3.1 Short PDU type

4.3.1.1 NOP command and response

NOP symbolizes a "no operation" command. Receiving this command, the slave shall do nothing but respond with a NOP-RSP1-PDU showing the latest alarm and status.

```
_NOP-CMD1-PDU        ::= _CMD1-PDU ( WITH COMPONENTS {..., cmd  (nop) } )
_NOP-RSP1-PDU        ::= _RSP1-PDU ( WITH COMPONENTS {..., rcmd (nop) } )
```

4.3.1.2 PRM_RD command and response

PRM_RD symbolizes a "parameter-read" command. Receiving this command, the slave shall read a specified parameter value into PRM_RD-RSP1-PDU.parameter field and shall respond with it

```
_PRM_RD-CMD1-PDU     ::= _CMD1-PDU ( WITH COMPONENTS
                              {..., cmd  (prm_rd),
                                pduBody (cmdBody _PRM_RD-CMD1Body) } )
_PRM_RD-CMD1Body     ::= SEQUENCE{ reserve1   OCTET STRING SIZE (3),
                             pNo   Unsigned16,
                             pSize Unsigned8,
                             reserve2   OCTET STRING SIZE (8) }
_PRM_RD-RSP1-PDU     ::= _RSP1-PDU ( WITH COMPONENTS
                              {..., rcmd (prm_rd),
                                pduBody (rspBody _PRM_RD-RSP1Body) } )
```

```
_PRM_RD-RSP1Body           ::= SEQUENCE { pNo   Unsigned16,
                                          pSize      Unsigned8,
                                          parameter OCTET STRING SIZE (8) }
```

- pNo field

pNo
This field shall contain the parameter number to read out.

The value range shall be from 0 to 65 535.

- pSize field

pSize
This field shall contain an octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter
This field shall contain the read data corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.3.1.3 PRM_WR command and response

PRM_WR symbolizes a "parameter-write" command. Receiving this command, the slave shall write a specified parameter value from PRM_WR-CMD1-PDU.parameter field and shall respond with the echo of it to tell completion of the command.

```
_PRM_WR-CMD1-PDU           ::= _CMD1-PDU ( WITH COMPONENTS
                                          {..., cmd (prm_wr),
                                           pduBody (cmdBody _PRM_WR-CMD1Body) } )
_PRM_WR-CMD1Body           ::= SEQUENCE { reserve   OCTET STRING SIZE (3),
                                          pNo Unsigned16,
                                          pSize Unsigned8,
                                          parameter OCTET STRING SIZE (8) }
_PRM_WR-RSP1-PDU           ::= _RSP1-PDU (WITH COMPONENTS
                                         {..., rcmd (prm_wr),
                                          pduBody (rspBody _PRM_WR-RSP1Body) } )
_PRM_WR-RSP1Body           ::= SEQUENCE { pNo Unsigned16,
                                          pSize Unsigned8,
                                          parameter OCTET STRING SIZE (8) }
```

- pNo field

pNo
This field shall contain the parameter number to write in.

The value range shall be from 0 to 65 535.

- pSize field

pSize
This field shall contain the octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter
This field shall contain the data to write corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.3.1.4 ID_RD command and response

ID_RD symbolizes a "device-ID-read" command. Receiving this command, the slave shall read a part of a specified item in the Device Information into ID_RD-RSP1-PDU.idData field and shall respond with it. The semantics and data types of the Device Information can be dependent on each device model, except idCode: 0, which shall specify the device model information.

```
_ID_RD-CMD1-PDU          ::= _CMD1-PDU (WITH COMPONENTS
                                 {..., cmd (id_rd),
                                  pduBody (cmdBody  _ID_RD-CMDBody) } )
_ID_RD-CMD1Body          ::= SEQUENCE { reserve1   OCTET STRING SIZE (3),
                                 idCode Unsigned8,
                                 idOffset Unsigned8,
                                 idSize Unsigned8,
                                 reserve2    OCTET STRING SIZE (8)}
_ID_RD-RSP1-PDU          ::= _RSP1-PDU (WITH COMPONENTS
                                 {..., rcmd (id_rd),
                                  pduBody (rspBody _ID_RD-RSPBody) } )
_ID_RD-RSP1Body          ::= SEQUENCE { idCode Unsigned8,
                                  idOffset Unsigned8,
                                  idSize Unsigned8,
                                  idData  OCTET STRING SIZE (8) }
```

- idCode field

idCode
This field shall contain the device-ID code to read out.

The value range shall be from 0 to 255.

  – '00'H: device model;
  – '01'H to 'FF'H: reserved.

- idOffset field

idOffset
This field shall contain the offset address of the device-ID to read out.

The value range shall be from 0 to 255.

- idSize field

idSize
This field shall contain the read octet size of the device-ID.

The value range shall be from 0 to 255.

- idData field

idData
This field shall contain the read data corresponding to the idCode. The data size, data structure and its meaning depend on the idCode.

4.3.1.5 CONFIG command and response

CONFIG symbolizes a "configure-device" command. Receiving this command, the slave shall activate a set of parameters on RAM updated with a PAR_WR command and shall respond to tell completion of the command.

```
_CONFIG-CMD1-PDU        ::= _CMD1-PDU (WITH COMPONENTS
                                 {..., cmd (config),
                                   pduBody (cmdBody _CONFIG-CMD1Body) } )
_CONFIG-CMD1Body        ::= SEQUENCE { reserve1   OCTET STRING SIZE (3),
                                config_mode Unsigned8 { pActive (0),
                                reserve2   OCTET STRING SIZE (10) }
_CONFIG-RSP1-PDU        ::= _RSP1-PDU (WITH COMPONENTS {..., rcmd (config) } )
```

- config_mode field

config_mode
This field shall contain the mode of device configuration.

This field shall be coded as data type Unsigned8 with the following value range.

- pActive ('00'H): to recalculate with the parameters and set up;
- '01'H to 'FF'H: reserved.

4.3.1.6 ALM_RD command and response

ALM_RD symbolizes an "alarm-read" command. Receiving this command, the slave shall read a set of alarm data that includes details of status of alarm and warning into ALM_RD-RSP1-PDU.alm_data field and shall respond with it. The semantics and data types of the alm_data are dependent on each device model.

```
_ALM_RD-CMD1-PDU        ::= _CMD1-PDU (WITH COMPONENTS
                                {..., cmd (alm_rd),
                                 pduBody (cmdBody _ALM_RD-CMD1Body) } )
_ALM_RD-CMD1Body        ::= SEQUENCE { reserve1   OCTET STRING SIZE (3),
                                 alm_rd_mode Unsigned8 { currentAlm (0)},
                                 reserve2    OCTET STRING SIZE (10) }
_ALM_RD-RSP1-PDU        ::= _RSP1-PDU (WITH COMPONENTS
                                {..., rcmd (alm_rd),
                                 pduBody (rspBody _ALM_RD-RSP1Body) } )
_ALM_RD-RSP1Body        ::= SEQUENCE { alm_data Unsigned8 { currentAlm (0)},
                                alm_data   OCTET STRING SIZE (10) }
```

- alm_rd_mode field

alm_rd_mode
This field shall contain the mode for the alarm reading.

This field shall be coded as data type Unsigned8 with the following value range.

- currentAlm ('00'H): to read out the current alarm/warning state;
- 01'H to 'FF'H: reserved.

- alm_data field

alm_data
This field shall contain the read alarm status corresponding to alm_rd_mode.

4.3.1.7 ALM_CLR command and response

ALM_CLR symbolizes an "alarm-clear" command. Receiving this command, the slave shall clear internal alarm and warning status and shall respond to tell completion of the command.

This command cannot work to solve a cause of alarm or warning. The command should be issued after solving the real causes or problems, to recover the normal status.

```
_ALM_CLR-CMD1-PDU        ::= _CMD1-PDU (WITH COMPONENTS
                                  {..., cmd (alm_clr),
                                   pduBody (cmdBody  _ALM_CLR-CMDBody) } )
_ALM_CLR-CMD1Body        ::= SEQUENCE { reserve1   OCTET STRING SIZE (3),
                                  alm_clr_mode Unsigned8 { cAlmClear(0) },
                                  reserve2   OCTET STRING SIZE (10) }
_ALM_CLR-RSP1-PDU        ::= _RSP1-PDU (WITH COMPONENTS
                                 {..., rcmd (alm_clr),
                                   pduBody (rspBody _ALM_CLR-RSP1Body) } )
_ALM_CLR-RSP1Body        ::= SEQUENCE { alm_clr_mode Unsigned8 { cAlmClear(0) },
                                  reserve   OCTET STRING SIZE (10) }
```

- alm_clr_mode field

alm_clr_mode
This field shall contain the mode for alarm clear.

This field shall be coded as data type Unsigned8 with the following value range.

– cAlmClear ('00'H): to clear the current alarm/warning status;
– '01'H to 'FF'H: reserved.

4.3.1.8 SYNC_SET command and response

SYNC_SET symbolizes a "set into synchronous-state" command. Receiving this command, the slave shall make a transition of slave PM (FDCPM-S) from S2: AsyncConnected state into S3: SyncConnected state. And receiving the response, the master shall make a similar transition of master PM (FDCPM-M) from S2: AsyncConnected state into S3: SyncConnected state, see 8.2.4 and 8.2.5 for more details.

```
_SYNC_SET-CMD1-PDU       ::= _CMD1-PDU (WITH COMPONENTS {..., cmd (sync_set) } )
_SYNC_SET-RSP1-PDU       ::= _RSP1-PDU (WITH COMPONENTS {..., rcmd (sync_set) } )
```

4.3.1.9 CONNECT command and response

After transmitting command and response, the master and the slave shall establish a FDC AR connection. At that time, some communication options or parameters can be specified with the PDUs. They shall make their transition of each PM; FDCPM-M and FDCPM-S, see 8.2.4 and 8.2.5 for more details.

```
_CONNECT-CMD1-PDU       ::= _CMD1-PDU (WITH COMPONENTS
                                {..., cmd (connect),
                                 pduBody (cmdBody _CONNECT-CMD1Body) } )
_CONNECT-CMD1Body       ::= SEQUENCE { reserve1   OCTET STRING SIZE (3),
                                ver Unsigned8,
                                com_mod SEQUENCE {
                                       reserve1    BIT STRING SIZE (1),
                                       syncmode  BIT STRING SIZE (1),
                                       dtmode      BIT STRING SIZE (2),
                                       reserve2    BIT STRING SIZE (3),
                                       subcmd     BIT STRING SIZE (1) },
                                 com_time Unsigned8,
                                 reserve2   OCTET STRING SIZE (8) }
_CONNECT-RSP1-PDU       ::= _RSP1-PDU (WITH COMPONENTS
                               {..., rcmd (connect),
                                pduBody (rspBody _CONNECT-RSP1Body) } )
_CONNECT-RSP1Body       ::= SEQUENCE { ver Unsigned8,
                                com_mod SEQUENCE {
                                      reserve1    BIT STRING SIZE (1),
                                      syncmode  BIT STRING SIZE (1),
                                      dtmode      BIT STRING SIZE (2),
                                      reserve2    BIT STRING SIZE (3),
                                      subcmd     BIT STRING SIZE (1) },
                                 com_time Unsigned8,
                                 reserve      OCTET STRING SIZE (10)}
```

- com_mod field

com_mod
This field shall contain several modes about communication in the connection.

syncmode
This bit field shall indicate the communication mode.

0: asynchronous communication state;
1: synchronous communication state.

dtmode
This bit field shall indicate the transfer mode

This field shall be with the following value range.

'00'H: single transfer mode;
'01'H: dual transfer mode;
'02'H to '03'H: reserved.

subcmd
This bit field shall indicate whether to use subcommand or not.

0: Subcommand field disabled;
1: Subcommand field enabled.

- com_time field

com_time
This field shall contain the communication cycle period in the form of a multiple of the transmission cycle period.

The value range shall be from 0 to 255.

4.3.1.10 DISCONNECT command and response

After transmitting command and response, the master and the slave shall release a FDC AR connection and specified communication options shall be reset. They shall make their transition of each PM; FDCPM-M and FDCPM-S, see 8.2.4 and 8.2.5 for more details.

```
_DISCONNECT-CMD1-PDU    ::= _CMD1-PDU (WITH COMPONENTS {..., cmd (disconnect) } )
_DISCONNECT-RSP1-PDU    ::= _RSP1-PDU (WITH COMPONENTS {..., rcmd  (disconnect) } )
```

4.3.1.11 PPRM_RD command and response

PRM_RD symbolizes a "PROM-parameter-read" command. Receiving this command, the slave shall read a specified parameter on E2PROM or non-volatile memory into PPRM_RD-RSP1-PDU.parameter field and shall respond with it.

```
_PPRM_RD-CMD1-PDU       ::= _CMD1-PDU (WITH COMPONENTS
                                       {..., cmd  (pprm_rd),
                                       pduBody (cmdBody _PPRM_RD-CMD1Body) } )
_PPRM_RD_CMD1Body       ::= SEQUENCE{ reserve1   OCTET STRING SIZE (3),
                                      pNo   Unsigned16,
                                      pSize Unsigned8,
                                      reserve2   OCTET STRING SIZE (8)}
_PPRM_RD-RSP1-PDU       ::= _RSP1-PDU (WITH COMPONENTS
                                      {..., rcmd  (pprm_rd),
                                       pduBody (rspBody _PPRM_RD-RSP1Body) } )
_PPRM_RD-RSP1Body       ::= SEQUENCE { pNo   Unsigned16,
                                       pSize        Unsigned8,
                                       parameter   OCTET STRING SIZE (8) }
```

- pNo field

pNo
This field shall contain the parameter number to read out.

The value range shall be from 0 to 65 535.

- pSize field

pSize
This field shall contain an octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter
This field shall contain the data corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.3.1.12 PPRM_WR command and response

PPRM_WR symbolizes a "PROM-parameter-write" command. Receiving this command, the slave shall write a specified parameter value from PPRM_WR-CMD1-PDU.parameter field to E2PROM or non-volatile memory and shall respond with the echo of it to tell completion of the command.

```
_PPRM_WR-CMD1-PDU       ::= _CMD1-PDU (WITH COMPONENTS
                                    {..., cmd (pprm_wr),
                                     pduBody (cmdBody _PPRM_WR-CMD1Body) } )
_PPRM_WR_CMD1Body       ::=  SEQUENCE { reserve   OCTET STRING SIZE (3),
                                    pNo Unsigned16,
                                    pSize Unsigned8,
                                    parameter OCTET STRING SIZE (8) }
_PPRM_WR-RSP1-PDU       ::= _RSP1-PDU (WITH COMPONENTS
                                   {..., rcmd (pprm_wr),
                                    pduBody (rspBody _PPRM_WR-RSP1Body) } )
_PPRM_WR-RSP1Body       ::= SEQUENCE { pNo Unsigned16,
                                    pSize Unsigned8,
                                    parameter OCTET STRING SIZE (8) }
```

- pNo field

pNo
This field shall contain the parameter number to write in.

The value range shall be from 0 to 65 535.

- pSize field

pSize
This field shall contain the octet size of the parameter.

The value range shall be from 0 to 255.

- parameter field

parameter
This field shall contain the data to write corresponding to pNo. The data size, data structure and its meaning depend on the pNo.

4.3.2 Long PDU type

4.3.2.1 NOP command and response

See 4.3.1.1.

```
_NOP-CMD2-PDU           ::= _CMD2-PDU (WITH COMPONENTS {..., cmd (nop) } )
_NOP-RSP2-PDU           ::= _RSP2-PDU (WITH COMPONENTS {..., rcmd (nop) } )
```

4.3.2.2 PRM_RD command and response

See 4.3.1.2.

```
_PRM_RD-CMD2-PDU          ::= _CMD2-PDU ( WITH COMPONENTS
                                             {..., cmd  (prm_rd),
                                              pduBody (cmdBody _PRM_RD-CMD1Body) } )
_PRM_RD-RSP2-PDU          ::= _RSP2-PDU ( WITH COMPONENTS
                                             {..., rcmd (prm_rd),
                                              pduBody (rspBody _PRM_RD-RSP1Body) } )
```

4.3.2.3 PRM_WR command and response

See 4.3.1.3.

```
_PRM_WR-CMD2-PDU          ::= _CMD2-PDU ( WITH COMPONENTS
                                             {..., cmd (prm_wr),
                                              pduBody (cmdBody _PRM_WR-CMD1Body) } )
_PRM_WR-RSP2-PDU          ::= _RSP2-PDU (WITH COMPONENTS
                                            {..., rcmd (prm_wr),
                                             pduBody (rspBody _PRM_WR-RSP1Body) } )
```

4.3.2.4 ID_RD command and response

See 4.3.1.4.

```
_ID_RD-CMD2-PDU           ::= _CMD2-PDU (WITH COMPONENTS
                                            {..., cmd (id_rd),
                                             pduBody (cmdBody  _ID_RD-CMDBody) } )
_ID_RD-RSP2-PDU           ::= _RSP2-PDU (WITH COMPONENTS
                                            {..., rcmd (id_rd),
                                             pduBody (rspBody _ID_RD-RSPBody) } )
```

4.3.2.5 CONFIG command and response

See 4.3.1.5.

```
_CONFIG-CMD2-PDU          ::= _CMD2-PDU (WITH COMPONENTS
                                             {..., cmd (config),
                                               pduBody (cmdBody _CONFIG-CMD1Body) } )
_CONFIG-RSP2-PDU          ::= _RSP2-PDU (WITH COMPONENTS {..., rcmd (config) } )
```

4.3.2.6 ALM_RD command and response

See 4.3.1.6.

```
_ALM_RD-CMD2-PDU        ::= _CMD2-PDU (WITH COMPONENTS
                                {…, cmd (alm_rd),
                                 pduBody (cmdBody _ALM_RD-CMD1Body) } )
_ALM_RD-RSP2-PDU        ::= _RSP2-PDU (WITH COMPONENTS
                                {…, rcmd (alm_rd),
                                 pduBody (rspBody _ALM_RD-RSP1Body) } )
```

4.3.2.7 ALM_CLR command and response

See 4.3.1.7.

```
_ALM_CLR-CMD2-PDU       ::= _CMD2-PDU (WITH COMPONENTS
                                 {…, cmd (alm_clr),
                                  pduBody (cmdBody  _ALM_CLR-CMDBody) } )
_ALM_CLR-RSP2-PDU       ::= _RSP2-PDU (WITH COMPONENTS
                                {…, rcmd (alm_clr),
                                  pduBody (rspBody _ALM_CLR-RSP1Body) } )
```

4.3.2.8 SYNC_SET command and response

See 4.3.1.8.

```
_SYNC_SET-CMD2-PDU      ::= _CMD2-PDU (WITH COMPONENTS {…, cmd (sync_set) } )
_SYNC_SET-RSP2-PDU      ::= _RSP2-PDU (WITH COMPONENTS {…, rcmd (sync_set) } )
```

4.3.2.9 CONNECT command and response

See 4.3.1.9.

```
_CONNECT-CMD2-PDU       ::= _CMD2-PDU (WITH COMPONENTS
                                 {…, cmd (connect),
                                  pduBody (cmdBody _CONNECT-CMD1Body) } )
_CONNECT-RSP2-PDU       ::= _RSP2-PDU (WITH COMPONENTS
                                {…, rcmd (connect),
                                 pduBody (rspBody _CONNECT-RSP1Body) } )
```

4.3.2.10 DISCONNECT command and response

See 4.3.1.10.

```
_DISCONNECT-CMD2-PDU    ::= _CMD2-PDU (WITH COMPONENTS {…, cmd (disconnect) } )
_DISCONNECT-RSP2-PDU    ::= _RSP2-PDU (WITH COMPONENTS {…, rcmd  (disconnect) } )
```

4.3.2.11 PPRM_RD command and response

See 4.3.1.11.

```
_PPRM_RD-CMD2-PDU        ::= _CMD2-PDU (WITH COMPONENTS
                                  {..., cmd (pprm_rd),
                                  pduBody (cmdBody _PPRM_RD-CMD1Body) } )
_PPRM_RD-RSP2-PDU        ::= _RSP2-PDU (WITH COMPONENTS
                                 {..., rcmd (pprm_rd),
                                  pduBody (rspBody _PPRM_RD-RSP1Body) } )
```

4.3.2.12 PPRM_WR command and response

See 4.3.1.12.

```
_PPRM_WR-CMD2-PDU        ::= _CMD2-PDU (WITH COMPONENTS
                                  {..., cmd (pprm_wr),
                                   pduBody (cmdBody _PPRM_WR-CMD1Body) } )
_PPRM_WR-RSP2-PDU        ::= _RSP1-PDU (WITH COMPONENTS
                                 {..., rcmd (pprm_wr),
                                  pduBody (rspBody _PPRM_WR-RSP1Body) } )
```

4.3.3 Enhanced PDU type

4.3.3.1 NOP command and response

See 4.3.1.1.

```
_NOP-CMD3-PDU            ::= _CMD3-PDU (WITH COMPONENTS{..., cmd (nop) } )
_NOP-RSP3-PDU            ::= _RSP3-PDU (WITH COMPONENTS {..., rcmd (nop) } )
```

4.3.3.2 PRM_RD command and response

See 4.3.1.2.

```
_PRM_RD-CMD3-PDU         ::= _CMD3-PDU (WITH COMPONENTS
                                  {..., cmd (prm_rd),
                                    pduBody (cmdBody  _PRM_RD-CMD3Body) }  )
_PRM_RD-CMD3Body         ::= SEQUENCE{ pNo   Unsigned16,
                                 pSize Unsigned8,
                                 reserve  OCTET STRING SIZE (1|9|25|41|57)}
_PRM_RD-RSP3-PDU         ::= _RSP3-PDU (WITH COMPONENTS
                                 {..., rcmd (prm_rd),
                                   pduBody (rspBody _PRM_RD-RSP3Body) }  )
_PRM_RD-RSP3Body         ::= SEQUENCE { pNo  Unsigned16,
                                 pSize       Unsigned8,
                                 reserve OCTET STRING SIZE(1),
                                  parameter OCTET STRING SIZE  (0|8|24|40|56)  }
```

4.3.3.3 PRM_WR command and response

See 4.3.1.3.

```
_PRM_WR-CMD3-PDU        ::= _CMD3-PDU (WITH COMPONENTS
                                 {..., cmd (prm_wr),
                                  pduBody (cmdBody _PRM_WR-CMD3Body) } )
_PRM_WR-CMD3Body        ::= SEQUENCE { pNo Unsigned16,
                                 pSize Unsigned8,
                                 reserve OCTET STRING SIZE(1),
                                 parameter OCTET STRING SIZE (0|8|24|40|56)  }
_PRM_WR-RSP3-PDU        ::= _RSP3-PDU (WITH COMPONENTS
                                {..., rcmd (prm_wr),
                                 pduBody (rspBody  _PRM_WR-RSP3Body) } )
_PRM_WR-RSP3Body        ::= SEQUENCE { pNo Unsigned16,
                                 pSize Unsigned8,
                                 reserve OCTET STRING SIZE(1),
                                 parameter OCTET STRING SIZE (0|8|24|40|56) }
```

4.3.3.4 ID_RD command and response

ID_RD symbolizes a "device-ID-read" command. Receiving this command, the slave shall read a part of a specified item in the Device Information into ID_RD-RSP3-PDU.idData field and shall respond with it. The semantics and data types of the Device Information are dependent on each device model, as shown in Annex B.

```
_ID_RD-CMD3-PDU         ::= _CMD3-PDU (WITH COMPONENTS
                                 {..., cmd (id_rd),
                                  pduBody (cmdBody _ID_RD-CMD3Body) } )
_ID_RD-CMD3Body         ::= SEQUENCE { idCode Unsigned8,
                                idOffcet Unsigned8,
                                idSize  Unsigned16 ,
                                reserve     OCTET STRINGSIZE (0|8|24|40|56)}
_ID_RD-RSP3-PDU         ::= _RSP3-PDU (WITH COMPONENTS
                                {..., rcmd (id_rd),
                                 pduBody (rspBody _ID_RD-RSP3Body) } )
_ID_RD-RSP3Body         ::= SEQUENCE { idCode Unsigned8,
                                idOffcet Unsigned8,
                                idSize  Unsigned16,
                                 idData  OCTET STRING SIZE (0|8|24|40|56) }
```

- idCode field

idCode
This field shall contain the device-ID code to read out.

The value range shall be from 0 to 255, see B.2.2.

– '00'H: reserved;
– '01'H: Vender ID code;
– '02'H: Device code or device model code;

- '03'H: Device version;
- '04'H: Version of Device Information file;
- '05'H: Number of extended addresses;
- '06'H: Serial number of product;
- '07'H to '0F'H: reserved;
- '10'H: Supported device profile code (primary);
- '11'H: Supported device profile version (primary);
- '12'H: Supported device profile code (secondary);
- '13'H: Supported device profile version (secondary);
- '14'H: Supported device profile code (tertiary);
- '15'H: Supported device profile version (tertiary);
- '16'H: Minimum transmission cycle;
- '17'H: Maximum transmission cycle;
- '18'H: Granularity of transmission cycle;
- '19'H: Minimum communication cycle;
- '1A'H: Maximum communication cycle;
- '1B'H: Number of the transmittable octets;
- '1C'H: Number of the transmitted octets (current setting);
- '1D'H: Device profile code (current setting);
- '1E'H to '1F'H: reserved;
- '20'H: Supported communication mode list;
- '21'H: MAC address;
- '22'H to '2F'H: reserved;
- '30'H: Supported main command list;
- '31'H to '37'H: reserved;
- '38'H: Supported sub command list;
- '39'H to '3F'H: reserved;
- '40'H: Supported common parameter list;
- '41'H to '7F'H: reserved;
- '80'H: Name of main device;
- '81'H to '8F'H: reserved;
- '90'H: Name of sub device 1;
- '91'H to '97'H: reserved;
- '98'H: Version of sub device 1;
- '99'H to '9F'H: reserved;
- 'A0'H: Name of sub device 2;
- 'A1'H to 'A7'H: reserved;
- 'A8'H: Version of sub device 2;
- 'A9'H to 'AF'H: reserved;
- 'B0'H: Name of sub device 3;
- 'B1'H to 'B7'H: reserved;
- 'B8'H: Version of sub device 3;
- 'B9'H to 'BF'H: reserved;

– 'C0'H to 'FF'H: reserved for vender specific information.

- idOffset field

idOffset
This field shall contain the offset address of the device-ID to read out.

The value range shall be from 0 to 255.

- idSize field

idSize
This field shall contain the read octet size of the device-ID.

The value range shall be from 0 to 255.

- idData field

idData
This field shall contain the read data corresponding to the idCode. The data size, data structure and its meaning depend on the idCode, see B.2.2.

4.3.3.5 CONFIG command and response

See 4.3.1.5. The code of "config_mode" shall be enhanced.

```
_CONFIG-CMD3-PDU        ::= _CMD3-PDU (WITH COMPONENTS
                                 {..., cmd (config),
                                  pduBody (cmdBody  _CONFIG-CMD3Body) } )
_CONFIG-CMD3Body        ::= SEQUENCE {config_mode Unsigned8 {pActive (0), pAllSave (1), pReset(2)},
                                reserve    OCTET STRING SIZE (3|11|27|43|59)}
_CONFIG-RSP3-PDU        ::= _RSP3-PDU (WITH COMPONENTS
                                {..., rcmd (config),
                                 pduBody (rspBody _CONFIG-RSP3Body) } )
_CONFIG-RSP3Body        ::= SEQUENCE {config_mode Unsigned8 {pActive (0), pAllSave (1), pReset(2)} ,
                                reserve  OCTET STRING  SIZE (3|11|27|43|59) }
```

- config_mode field

config_mode
This field shall contain the mode of device configuration.

This field shall be coded as data type Unsigned8 with the following value range.

– pActive ('00'H): to recalculate with the parameters and set up;
– pAllSave ('01'H): to write parameters into nonvolatile memory with batch;
– pReset ('02'H): to restore to the factory setting of parameters;
– '03'H to 'FF'H: reserved.

4.3.3.6 ALM_RD command and response

See 4.3.1.6. The code of "alarm_rd_mode" and corresponding alm_data shall be enhanced.

```
_ALM_RD-CMD3-PDU          ::= _CMD3-PDU (WITH COMPONENTS
                                          {..., cmd (alm_rd),
                                           pduBody (cmdBody _ALM_RD-CMD3Body) }  )
_ALM_RD-CMD3Body          ::=  SEQUENCE{ alm_rd_mode Unsigned16
                                           { currentAlm (0), historyAlm (1),
                                               cAlmDetail (2), hAlmDetail (3)} ,
                                          alm_index Unsigned16 (0..11) ,
                                          reserve    OCTET STRING SIZE (0|8|24|40|56) }
_ALM_RD-RSP3-PDU          ::= _RSP3-PDU (WITH COMPONENTS
                                         {..., rcmd (alm_rd),
                                           pduBody (cmdBody  _ID_RD-RSP3Body) } )
_ALM_RD-RSP3Body          ::= SEQUENCE{ alm_rd_mode Unsigned16
                                            { currentAlm (0), historyAlm (1),
                                              cAlmDetail (2), hAlmDetail (3)} ,
                                         alm_index Unsigned16 (0..11) ,
                                         alm_data   OCTET STRING SIZE (8|24|40|56) }
```

- alm_rd_mode field

alm_rd_mode
This field shall contain the mode for the alarm reading.

This field shall be coded as data type Unsigned8 with the following value range.

- currentAlm ('00'H): to read out the current alarm/warning state;
- historyAlm ('01'H): to read out the alarm history;
- cAlmDetail ('02'H): to retrieve details of individual current alarm/warning information;
- hAlmDetail ('03'H): to retrieves details of individual alarm history information;
- '04'H to 'FF'H: reserved.

- alm_index field

alm_index
This field shall contain the index for reading point of alarm/warning.

The value range shall be from 0 to 11.

- alm_data field

alm_data
This field shall contain the read alarm status corresponding to alm_rd_mode.

4.3.3.7 ALM_CLR command and response

See 4.3.1.7. The code of "alm_clr_mode" shall be enhanced.

```
_ALM_CLR-CMD3-PDU       ::= _CMD3-PDU (WITH COMPONENTS
                                 {..., cmd (alm_clr),
                                  pduBody (cmdBody _ALM_CLR-CMD3Body) } )
_ALM_CLR-CMD3Body       ::= SEQUENCE { alm_clr_mode Unsigned16 { cAlmClear (0), hAlmClear(1)} ,
                                 reserve OCTET STRING SIZE (2|10|26|42|58)}
_ALM_CLR-RSP3-PDU       ::= _RSP3-PDU (WITH COMPONENTS
                                 {..., rcmd (alm_clr),
                                   pduBody (rspBody  _ALM_CLR-RSP3Body) } )
_ALM_CLR-RSP3Body       ::= SEQUENCE { alm_clr_mode Unsigned16 { cAlmClear (0), hAlmClear(1)},
                                reserve OCTET STRING SIZE (2|10|26|42|58)}
```

- alm_clr_mode field

alm_clr_mode
This field shall contain the mode for alarm clear.

This field shall be coded as data type Unsigned8 with the following value range.

– cAlmClear ('00'H): to clear the current alarm/warning status;
– hAlmClear ('01'H): to clear the alarm history;
– '02'H to 'FF'H: reserved.

4.3.3.8 SYNC_SET command and response

See 4.3.1.8.

```
_SYNC_SET-CMD3-PDU      ::= _CMD3-PDU (WITH COMPONENTS{..., cmd (sync_set) } )
_SYNC_SET-RSP3-PDU      ::= _RSP3-PDU (WITH COMPONENTS {..., rcmd (sync_set) } )
```

4.3.3.9 CONNECT command and response

See 4.3.1.9. The field of "profile_type" shall be enhanced.

```
_CONNECT-CMD3-PDU       ::= _CMD3-PDU (WITH COMPONENTS
                                 {..., cmd (connect),
                                  pduBody (cmdBody  _CONNECT-CMD3Body) } )
_CONNECT-CMD3Body       ::= SEQUENCE { ver Unsigned8,
                                  com_mod SEQUENCE {
                                        reserve1   BIT STRING SIZE (1),
                                        syncmode   BIT STRING SIZE (1),
                                        dtmode     BIT STRING SIZE (2),
                                        reserve2   BIT STRING SIZE (3),
                                        subcmd     BIT STRING SIZE (1) },
                                  com_time Unsigned8
                                  profile_type Unsigned8
                                  reserve     OCTET STRING SIZE (0|8|24|40|56) }
```

```
_CONNECT-RSP3-PDU        ::= _RSP3-PDU (WITH COMPONENTS
                                  {..., rcmd (connect),
                                   pduBody (rspBody _CONNECT-RSP3Body) } )
_CONNECT-RSP3Body        ::= SEQUENCE { ver Unsigned8,
                                   com_mod SEQUENCE {
                                         reserve1    BIT STRING SIZE (1),
                                         syncmode  BIT STRING SIZE (1),
                                         dtmode      BIT STRING SIZE (2),
                                         reserve2    BIT STRING SIZE (3),
                                         subcmd     BIT STRING SIZE (1) },
                                    com_time Unsigned8
                                    profile_type Unsigned8
                                    reserve      OCTET STRING SIZE (0|8|24|40|56)}
```

- profile_type field

profile_type
This field shall contain the device profile code to be used. As for the device profile, see 4.4 and Annex A.

The value range shall be from 0 to 255.

4.3.3.10 DISCONNECT command and response

See 4.3.1.10.

```
_DISCONNECT-CMD3-PDU    ::= _CMD3-PDU (WITH COMPONENTS {..., cmd (disconnect) } )
_DISCONNECT-RSP3-PDU    ::= _RSP3-PDU (WITH COMPONENTS {..., rcmd (disconnect) } )
```

4.3.3.11 PPRM_RD command and response

See 4.3.1.11.

```
_PPRM_RD-CMD3-PDU        ::= CMD3-PDU (WITH COMPONENTS
                                  {..., cmd (pprm_rd),
                                   pduBody (cmdBody  _PPRM_RD-CMD3Body) } )
_PPRM_RD_CMD3Body        ::= SEQUENCE{ pNo   Unsigned16,
                                  pSize Unsigned8
                                  reserve     OCTET STRING SIZE (1|9|25|41|57)}
_PPRM_RD-RSP3-PDU        ::= _RSP3-PDU (WITH COMPONENTS
                                  {..., rcmd (pprm_rd),
                                   pduBody (rspBody  _PPRM_RD-RSP3Body) } )
_PPRM_RD-RSP3Body        ::= SEQUENCE { pNo   Unsigned16,
                                  pSize          Unsigned8,
                                  reserve        OCTET STRING SIZE(1),
                                  parameter     OCTET STRING SIZE (0|8|24|40|56) }
```

4.3.3.12 PPRM_WR command and response

See 4.3.1.12.

```
_PPRM_WR-CMD3-PDU       ::= _CMD3-PDU (WITH COMPONENTS
                               {..., cmd (pprm_wr),
                                pduBody (cmdBody _PPRM_WR-CMD3Body) } )
_PPRM_WR_CMD3Body       ::= SEQUENCE { pNo Unsigned16,
                               pSize Unsigned8,
                               reserve OCTET STRING SIZE(1),
                               parameter OCTET STRING SIZE (0|8|24|40|56) }
_PPRM_WR-RSP3-PDU       ::= _RSP3-PDU (WITH COMPONENTS
                               {..., rcmd (pprm_wr),
                                pduBody (rspBody  _PPRM_WR-RSP3Body) } )
_PPRM_WR-RSP3Body       ::= SEQUENCE { pNo Unsigned16,
                               pSize Unsigned8,
                               reserve OCTET STRING SIZE(1),
                               parameter OCTET STRING SIZE (0|8|24|40|56) }
```

4.3.3.13 MEM_RD command and response

MEM_RD symbolizes a "memory-read" command. Receiving this command, the slave shall read a chunk of specified virtual memory into MEM_RD-RSP3-PDU.data field and shall respond with it.

As for virtual memory, see Annex A.

```
_MEM_RD-CMD3-PDU        ::= _CMD3-PDU (WITH COMPONENTS
                               {..., cmd (mem_rd),
                                pduBody (cmdBody  _MEM_RD-CMD3Body) } )
_MEM_RD-CMD3Body        ::=SEQUENCE { reserve OCTET STRING,
                              mMode      SEQUENCE {
                                                   data_type   BIT STRING SIZE (4),
                                                   mode        BIT STRING SIZE (4) },
                              mSize      Unsigned16,
                              mAddress   Unsigned32,
                              reserve    OCTET STRING SIZE (4|20|36|52)}
_MEM_RD-RSP3-PDU        ::= _RSP3-PDU (WITH COMPONENTS
                              {..., rcmd (mem_rd),
                                pduBody (rspBody  _MEM_RD-RSP3Body) } )
_MEM_RD-RSP3Body        ::=SEQUENCE { reserve OCTET STRING,
                              mMode      SEQUENCE {
                                                   data_type   BIT STRING SIZE (4),
                                                   mode        BIT STRING SIZE (4) },
                               mSize      Unsigned16,
                               mAddress   Unsigned32,
                               data       OCTET STRING SIZE (4|20|36|52) }
```

- mMode field

mMode
This field shall contain the mode for the memory reading-out.

data_type
This bit field contains a type of the memory as source to read out.

'00'H: reserved;

'01'H: Volatile memory;

'02'H: Non-volatile memory;

'03'H to 'FF'H: reserved.

mode

This bit field contains the data type of the specified data on memory to read out.

'00'H: reserved;

'01'H: Integer8;

'02'H: Integer16;

'03'H: Integer32;

'04'H: Integer64;

'05'H to 'FF'H: reserved.

- mSize field

mSize
This field shall contain the number of data on memory to read out.

The value range shall be from 0 to 20.

- mAddress field

mAddress
This field shall contain the start address of memory to read out.

The value range shall be from 0 to 'FFFF FFFF'H.

- data field

data
This field shall contain the read data with the specified mMode, mSize and mAddress.

4.3.3.14 MEM_WR command and response

MEM_WR symbolizes a "memory-write" command. Receiving this command, the slave shall write a chunk of specified virtual memory value from MEM_WR-CMD3-PDU.data field to the memory and shall respond with the echo of it to tell completion of the command.

As for virtual memory, see Annex A.

```
_MEM_WR-CMD3-PDU        ::= _CMD3-PDU (WITH COMPONENTS
                                  {..., cmd (mem_wr),
                                    pduBody (cmdBody  _MEM_RD-CMD3Body) } )
_MEM_WR-CMD3Body        ::=SEQUENCE { reserve OCTET STRING,
                                 mMode      SEQUENCE {
                                                      data_type   BIT STRING SIZE (4),
                                                      mode        BIT STRING SIZE (4) },
                                  mSize       Unsigned16,
                                  mAddress  Unsigned32,
                                  data          OCTET STRING SIZE (4|20|36|52)  }
_MEM_WR-RSP3-PDU        ::= _RSP3-PDU (WITH COMPONENTS
                                 {..., rcmd (mem_wr),
                                   pduBody (rspBody  _MEM_RD-RSP3Body) } )
_MEM_WR-RSP3Body        ::=SEQUENCE { reserve OCTET STRING,
                                 mMode      SEQUENCE {
                                                         data_type   BIT STRING SIZE (4),
                                                         mode         BIT STRING SIZE (4) },
                                 mSize       Unsigned16,
                                 mAddress  Unsigned32
                                 data          OCTET STRING SIZE (4|20|36|52)  }
```

- mMode field

mMode
This field shall contain the mode for the memory writing-in.

data_type
This bit field shall contain a type of the memory as destination to write in.

'00'H: reserved;

'01'H: Volatile memory;

'02'H: Non-volatile memory;

'03'H to 'FF'H: reserved.

mode

This bit field shall contain the data type of the specified data on memory to write in.

'00'H: reserved;

'01'H: Integer8;

'02'H: Integer16;

'03'H: Integer32;

'04'H: Integer64;

'05'H to 'FF'H: reserved.

- mSize field

mSize
This field shall contain the number of data on memory to write in.

The value range shall be from 0 to 20.

- mAddress field

mAddress
This field shall contain the start address of memory to write in.

The value range shall be from 0 to 'FFFF FFFF'H.

- data field

data
This field shall contain the data to write with the specified mMode, mSize and mAddress.

4.3.4 SubCommand PDU type

4.3.4.1 NOP sub command and response

See 4.3.1.1.

```
_NOP-SUBCMD-PDU          ::= _USUBCMD-PDU (WITH COMPONENTS {..., subcmd (nop) } )
_NOP-SUBRSP-PDU          ::= _USUBRSP-PDU (WITH COMPONENTS {..., rsubcmd (nop) } )
```

4.3.4.2 PRM_RD sub command and response

See 4.3.1.2.

```
_PRM_RD-SUBCMD-PDU       ::= _USUBCMD-PDU (WITH COMPONENTS
                                              {..., subcmd (prm_rd),
                                               pduBody (cmdBody   _PRM_RD-CMD1Body) } )
_PRM_RD-SUBRSP-PDU       ::= _USUBRSP-PDU (WITH COMPONENTS
                                              {..., rsubcmd (prm_rd),
                                                pduBody (rspBody _PRM_RD-RSP1Body) } )
```

4.3.4.3 PRM_WR sub command and response

See 4.3.1.3.

```
_PRM_WR-SUBCMD-PDU       ::= _USUBCMD-PDU (WITH COMPONENTS
                                              {..., subcmd (prm_wr),
                                               pduBody (cmdBody  _PRM_WR-CMD1Body) } )
_PRM_WR-SUBRSP-PDU       ::= _USUBRSP-PDU (WITH COMPONENTS
                                              {..., rsubcmd (prm_wr),
                                                pduBody (rspBody  _PRM_WR-RSP1Body) } )
```

4.3.4.4 ALM_RD sub command and response

See 4.3.1.5.

```
_ALM_RD-SUBCMD-PDU       ::= _USUBCMD-PDU (WITH COMPONENTS
                                              {..., subcmd (alm_rd),
                                                pduBody (cmdBody  _ALM_RD-CMD1Body) } )
_ALM_RD-SUBRSP-PDU       ::= _USUBRSP-PDU (WITH COMPONENTS
                                              {..., rsubcmd (alm_rd),
                                                pduBody (rspBody  _ALM_RD-RSP1Body) }  )
```

4.3.4.5 PPRM_RD sub command and response

See 4.3.1.11.

```
_PPRM_RD-SUBCMD-PDU    ::= _USUBCMD-PDU (WITH COMPONENTS
                                          {..., subcmd (pprm_rd),
                                           pduBody (cmdBody _PPRM_RD-CMD1Body) } )
_PPRM_RD-SUBRSP-PDU    ::= _USUBRSP-PDU (WITH COMPONENTS
                                          {..., rsubcmd (pprm_rd),
                                           pduBody (rspBody  _PPRM_RD-RSP1Body) } )
```

4.3.4.6 PPRM_WR sub command and response

See 4.3.1.12.

```
_PPRM_WR-SUBCMD-PDU    ::= _USUBCMD-PDU (WITH COMPONENTS
                                           {..., subcmd (pprm_wr),
                                             pduBody (cmdBody _PPRM_WR-CMD1Body) } )
_PPRM_WR-SUBRSP-PDU    ::= _USUBRSP-PDU (WITH COMPONENTS
                                          {..., rsubcmd (pprm_wr),
                                            pduBody (rspBody  _PPRM_WR-RSP1Body) } )
```

4.3.5 Short PDU type II

4.3.5.1 NOP command and response

NOP symbolizes a "no operation" command. Receiving this command, the slave shall do nothing but respond with a NOP-RSP4-PDU showing the latest alarm and status.

```
_NOP-CMD4-PDU          ::= _CMD4-PDU ( WITH COMPONENTS {..., cmd2  (nop) } )
_NOP-RSP4-PDU          ::= _RSP4-PDU ( WITH COMPONENTS {..., rcmd2 (nop) } )
```

4.3.5.2 ECHO command and response

ECHO symbolizes an "echo back" command. Receiving this command, the C2 master and the slave shall set the data in the ECHO-CMD4-PDU.echodata field to the ECHO-RSP4-PDU.echodata field and respond to the C1 master.

```
_ECHO-CMD4-PDU         ::= _CMD4-PDU ( WITH COMPONENTS
                                      {..., cmd  (echo),
                                       pduBody (cmdBody _ECHO-CMD4Body) } )
_ECHO-CMD4Body         ::= SEQUENCE{ echodata OCTET STRING SIZE (14) }
_ECHO-RSP4-PDU         ::= _RSP4-PDU ( WITH COMPONENTS
                                      {..., rcmd2 (echo),
                                       pduBody (rspBody _ECHO-RSP4Body) } )
_ECHO-RSP4Body         ::= SEQUENCE { echodata OCTET STRING SIZE (14) }
```

- echodata field

echodata
This field shall contain the echo data.

4.3.5.3 CHECK_ID command and response

CHECK_ID symbolizes a "Check device-ID" command. Receiving this command, the C2 master and the slave shall set the device information in the CHECK_ID-RSP4-PDU.idData field and respond to the C1 master.

```
_CHECK_ID-CMD4-PDU        ::= _CMD4-PDU ( WITH COMPONENTS
                                          {..., cmd (check_id),
                                           pduBody (cmdBody _CHECK_ID-CMD4Body) } )
_CHECK_ID-CMD4Body        ::= SEQUENCE { reserve   OCTET STRING SIZE (14) }
_CHECK_ID-RSP4-PDU        ::= _RSP4-PDU (WITH COMPONENTS
                                         {..., rcmd (check_id),
                                          pduBody (rspBody _CHECK_ID-RSP4Body) } )
_CHECK_ID-RSP4Body        ::= SEQUENCE { idData OCTET STRING SIZE(12),
                                          reserve OCTET STRING SIZE(2) }
```

- idData field

idData

This field shall contain the device information of the slave that has been read.

4.3.5.4 SET_COM command and response

SET_COM symbolizes a "Set Communication Parameter" command. Receiving this command, the C2 master and the slave shall set the data of SET_COM-CMD4-PDU.cmdBody to SET_COM-RSP4-PDU.rspBody and respond to the C1 master. After this, the C2 master and the slave change the communication setting of their own station according to SET_COM-CMD4-PDU.com_setting. This SET_COM command can be used only in the event driven mode. If this SET_COM command is received in the cyclic communication mode, the C2 master and the slave shall respond an error to the C1 master.

```
_SET_COM-CMD4-PDU         ::= _CMD4-PDU ( WITH COMPONENTS
                                          {..., cmd (set_com),
                                           pduBody (cmdBody _SET_COM-CMD4Body) } )
_SET_COM-CMD4Body         ::= SEQUENCE { reserve1   OCTET STRING SIZE (1),
                                          com_setting SEQUENCE {
                                            line_code BIT STRING SIZE (4),
                                            baud_rate BIT STRING SIZE (4) },
                                          reserve2 OCTET STRING (2),
                                          tcc_pos Unsigned8.
                                          tcc_mul Unsigned8,
                                          reserve3 OCTET STRING SIZE (8) }
_SET_COM-RSP4-PDU         ::= _RSP4-PDU (WITH COMPONENTS
                                         {..., rcmd (set_com),
                                          pduBody (rspBody _SET_COM-RSP4Body) } )
```

```
_SET_COM-RSP4Body        ::= SEQUENCE { reserve1   OCTET STRING SIZE (1),
                                        com_setting SEQUENCE {
                                          line_code BIT STRING SIZE (4),
                                          baud_rate BIT STRING SIZE (4) },
                                        reserve2 OCTET STRING (2),
                                        tcc_pos Unsigned8.
                                        tcc_mul Unsigned8,
                                        reserve3 OCTET STRING SIZE (8) }
```

- com_setting field

com_setting
This field shall contain the information on the baud rates and the encoding methods for setting.

line_code[a]
This bit field indicates the encoding rule.

'0'H: Manchester encoding;

'1'H: NRZI encoding;

Other: reserved.

baud_rate[a]
This bit field indicates the baud rate.

'1'H: 4Mbit/s;

'2'H: 8Mbit/s;

'4'H: 16Mbit/s;

'8'H: 24Mbit/s;

'9'H: 32Mbit/s;

Other: reserved.

The setting range of baud_rate depends on line_code and shall be within the following range.

– Manchester encoding: 4Mbit/s | 8Mbit/s

– NRZI encoding: 16Mbit/s | 24Mbit/s | 32Mbit/s

- tcc_pos field

tcc_pos

This field indicates the individual transmission position number in the communication with multiple transmission cycles. The setting range shall equal to or be less than the setting of tcc_mul field.

- tcc_mul field

tcc_mul
This field indicates the individual transmission cycle multiple in the communication with multiple transmission cycles. The setting shall be an integer multiple of the basic transmission cycle (Tcyc) and the range shall be from 1 to 256.

4.3.5.5 SET_ADDR command and response

SET_ADDR symbolizes a "Set Address" command. Receiving this command, the C2 master and the slave shall set the data of SET_ADDR-CMD4-PDU.cmdBody to SET_ADDR-RSP4.rspBody and respond to the C1 master. After this, the C2 master and the slave change the node address of its own station according to SET_ADDR-CMD4-PDU.addr. This SET_ADDR command can be used only in the event driven mode. If this SET_ADDR command is received in the cyclic communication mode, the C2 master and the slave shall respond an error to the C1 master.

```
_SET_ADDR-CMD4-PDU      ::= _CMD4-PDU ( WITH COMPONENTS
                                         {…, cmd (set_addr),
                                          pduBody (cmdBody _SET_ADDR-CMD4Body) } )
_SET_ADDR-CMD4Body      ::= SEQUENCE { reserve1   OCTET STRING SIZE (1),
                                          addr Ungined8,
                                          function_setting SEQUENCE {
                                              repeat BIT STRING SIZE (1),
                                              reserve2 BIT STRING SIZE (7),
                                         reserve3   OCTET STRING SIZE (11) }
_SET_ADDR-RSP4-PDU      ::= _RSP4-PDU (WITH COMPONENTS
                                        {…, rcmd (set_addr),
                                         pduBody (rspBody _SET_COM-RSP4Body) } )
_SET_ADDR-RSP4Body      ::= SEQUENCE { reserve1   OCTET STRING SIZE (1),
                                         addr Ungined8,
                                         function_setting SEQUENCE {
                                              repeat BIT STRING SIZE (1),
                                              reserve2 BIT STRING SIZE (7),
                                         reserve3   OCTET STRING SIZE (11) }
```

- addr field

addr
This field shall contain the address to set.

- function field

function

This field indicates the setting of repeat function of a transmission frame.

repeat
This bit field indicates active / not active of the repeat function.

0: repeat function is not active;

1: repeat function is active.

4.3.5.6 MEM_RD command and response

MEM_RD symbolizes a "Memory Read" command. Receiving this command, the C2 master and the slave shall set the data for the length bytes from the memory address (addrHigh and addrLow) setting in MEM_RD-CMD4-PDU.cmdBody to the MEM_RD-RSP4-PDU.rdData field and respond to the C1 master.

```
_MEM_RD-CMD4-PDU        ::= _CMD4-PDU ( WITH COMPONENTS
                                 {..., cmd (mem_rd),
                                  pduBody (cmdBody _MEM_RD-CMD4Body) } )
_MEM_RD-CMD4Body        ::= SEQUENCE { len_addrH   SEQUENCE{
                                   addrHigh BIT STRING SIZE (4),
                                   length BIT STRING SIZE (4) },
                                 addrLow Unsigned8,
                                 reserve   OCTET STRING SIZE (12) }
_MEM_RD-RSP4-PDU        ::= _RSP4-PDU (WITH COMPONENTS
                                {..., rcmd (mem_rd),
                                 pduBody (rspBody _MEM_RD-RSP4Body) } )
_MEM_RD-RSP4Body        ::= SEQUENCE { len_addrH   SEQUENCE{
                                   addrHigh BIT STRING SIZE (4),
                                   length BIT STRING SIZE (4) },
                                 addrLow Unsigned8,
                                 rdData OCTET STRING SIZE (8),
                                 reserve   OCTET STRING SIZE (4) }
```

- len_addrH field

len_addrH
This field shall contain the byte size of the data to read and the high-order bit of the data to read.

addrHigh
This bit field indicates the high-order address of the data to read.

len
This bit field indicates the byte size of the data to read.

The range shall be from 1 byte to 8 bytes.

- addrL field

addrLow
This field shall contain the low-order address of the data to read.

- rdData field

rdData
This field shall contain the data that has been read.

4.3.5.7 MEM_WR command and response

MEM_WR symbolizes a "Memory Write" command. Receiving this command, the C2 master and the slave shall write the data that is set to wrData in the length byte area from the memory address (addrHigh and addrLow) in the MEM_WR-CMD4-PDU.cmdBody setting, and set the data that has been written to the MEM_RD-RSP4-PDU.rdData field, and respond to the C1 master.

```
_MEM_WR-CMD4-PDU        ::= _CMD4-PDU ( WITH COMPONENTS
                                        {..., cmd (mem_wr),
                                         pduBody (cmdBody _MEM_WR-CMD4Body) } )
_MEM_WR-CMD4Body        ::= SEQUENCE { len_addrH   SEQUENCE{
                                          length BIT STRING SIZE (4),
                                          addrHigh BIT STRING SIZE (4) },
                                        addrLow Unsigned8,
                                        wrData   OCTET STRING SIZE (8),
                                        reserve OCTET STRING SIZE (4) }
_MEM_WR-RSP4-PDU        ::= _RSP4-PDU (WITH COMPONENTS
                                       {..., rcmd (mem_rd),
                                        pduBody (rspBody _MEM_RD-RSP4Body) } )
_MEM_WR-RSP4Body        ::= SEQUENCE { len_addrH   SEQUENCE{
                                          length BIT STRING SIZE (4),
                                          addrHigh BIT STRING SIZE (4) },
                                        addrLow Unsigned8,
                                        wrData   OCTET STRING SIZE (8),
                                        reserve OCTET STRING SIZE (4) }
```

- len_addrH field

len_addrH
This field shall contain the byte size of the data to write and the high-order bit of the data to write.

> **len**
> This bit field indicates the byte size of the data to write.
>
> The range shall be from 1 byte to 8 bytes.
>
> **addrHigh**
> This bit field shall contain the high-order address of the data to write.

- addrL field

addrLow
This field shall contain the low-order address of the data to write.

- wrData field

wrData
This field shall contain the data to write.

4.3.5.8 CHG_MST command and response

CHG_MST symbolizes a "Change Master" command. Receiving this command from the C1 master, the C2 master shall respond on "asynchronous command processing request: Exist / Not exist" in the C2 master to the C1 master. If the C2 master responds "asynchronous command processing request: Exist", the C2 master executes the asynchronous command processing with the slave that the C2 master controls. If the C2 master responds "asynchronous command processing request: Exist", the C1 master cyclically sends the CHG_CMD command to the C2 master and repeats this until t the C2 master responds "asynchronous command processing request: Not exist". After completing the asynchronous command processing, the C2 master receives the CHG_CMD command from the C1 master and responds "asynchronous command processing request: Not exist". This CHG_MST command can be used only in the event driven mode. If this CHG_MST command is received in the cyclic communication mode, the C2 master and the slave shall respond a to the C1 master.

```
_CHG_MST-CMD4-PDU        ::= _CMD4-PDU ( WITH COMPONENTS {..., cmd (chg_mst)} )
_CHG_MST-RSP4-PDU        ::= _RSP4-PDU (WITH COMPONENTS {..., rcmd (chg_mst),
                                         cmdExist Unsigned8,
                                         reserve OCTET STRING SIZE(13) }
```

- cmdExist field

cmdExist
This field indicates the sending data Exist / Not exist.

– '01'H: asynchronous command processing request: Exist
– '02'H: asynchronous command processing request: Not exist

4.3.5.9 CHK_MSG_REQ command and response

CHK_MSG_REQ symbolizes a "Check Message transmission Request" command. Receiving this command from the C1 master, the C2 master shall respond "asynchronous command processing request: Exist / Not exist" in the C2 master, and if it responds "asynchronous command processing request: Exist", it shall respond the asynchronous command data (slave address to send and command data to send) to the C1 master. If the C1 master receives the response of "asynchronous command processing request: Exist" from the C2 master, the C1 master executes the processing of asynchronous command data contained in the response. The C2 master monitors the INPUT data frame sent from the slave (the response data to the asynchronous command that has been sent to the C1 master) and loads the matched INPUT data frame into the C2 master. This CHK_MSG_REQ command can be used only in the cyclic communication mode. If this CHK_MSG_REQ command is received in the event driven mode, the C2 master shall respond an error to the C1 master.

```
_CHG_MST-CMD4-PDU        ::= _CMD4-PDU ( WITH COMPONENTS {..., cmd (chk_msg_req)} )
_CHG_MST-RSP4-PDU        ::= _RSP4-PDU (WITH COMPONENTS
                                        {..., rcmd (chk_msg_req),
                                         slvAddr Unsigned8,
                                         dataNum Unsined8,
                                         sndData OCTET STRING SIZE(12) }
```

- slvAddr field

slvAddr
This field indicates the address to send the message. The range shall be from '10'H to '1E'H.

- dataNum field

dataNum
This field indicates the number of bytes of the message to send. The range shall be from 1 to 12.

- sndData field

sndData
This field indicates the data of the message to send.

4.4 Device profile

A field device can provide some sets of field-device-dependent PDUs and their CMDCodes for FDC service, called field device profiles. The user of an FDC master can decide which of them he should choose for FDC communication when the connection of FDC protocol is established.

The device Profile system and related information are shown in Annex A.

5 Transfer syntax

5.1 Concepts

The Type 24 FAL is a technology for a time critical application. Therefore, it is important to make the size of encoded data compact, and to enable to encode and decode in a simple manner.

Consequently, the encoding and decoding processes shall be defined as such specific to the Type 24 FAL, and require neither any commonality with the other protocol type nor versatility. The data format and the encode rule of APDU shall be pre-defined for entities that transfer the data, and require no presentation layer service.

In the encoding rule of the Type 24 FAL, the tag (data type code) and the data length need not be encoded and the only values of each data types defined with ASN.1 abstract syntax shall be encoded into a data row.

The length of PDU shall be fixed in case of a FDC service where the time factor is particularly strict. For a MSG service where the time factor is not so strict, PDU with variable length can be transferred. In this case, a data field that specifies the data length should be defined in the abstract syntax explicitly.

The order of the bits flow on the transmission path should be defined by the lower layer.

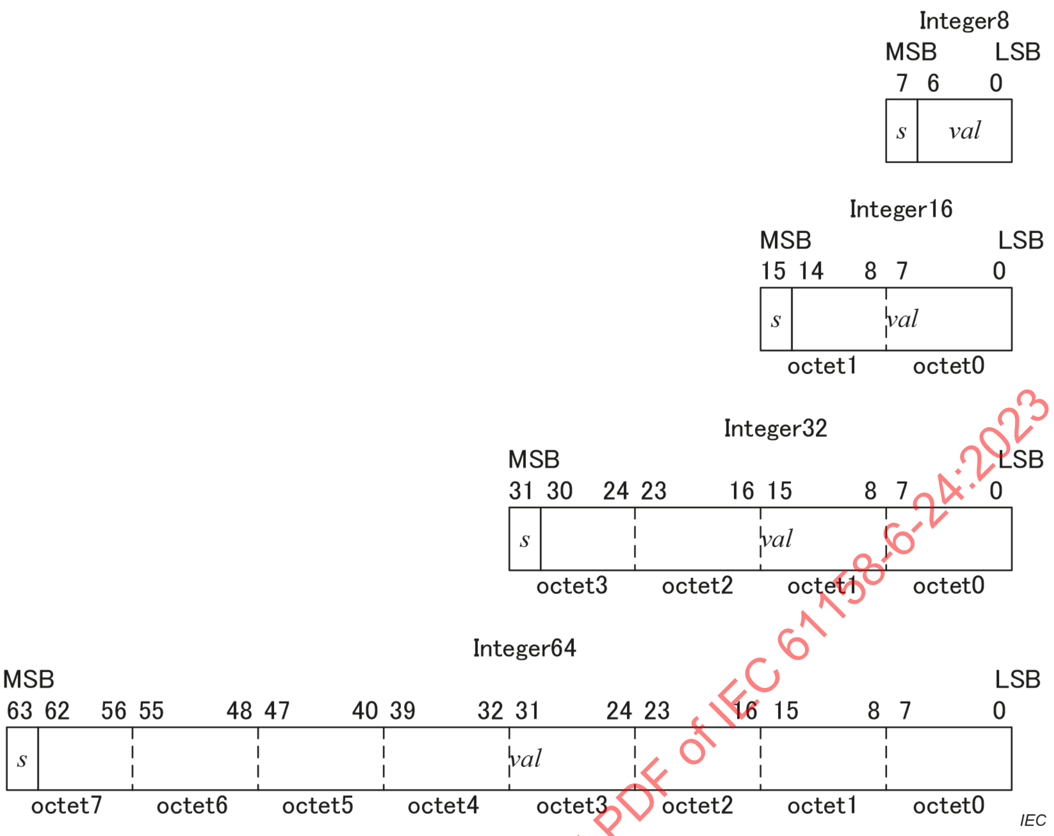
5.2 Encode rules

5.2.1 INTEGER and its subtypes

INTEGER and its derivatives, Integer8, Integer16, Integer32, Integer64, Unsigned8, Unsigned16, Unsigned32 and Unsigned64 shall encode and transfer only the data value of their octet size.

The encoded form of Integer8, Integer16, Integer32 and Integer64 shall consist of a sign bit:s on the most significant bit (see Figure 2). The val shall contain the value itself when it is 0 or a positive number (the sign bit is '0'B). When val is a negative number (the sign bit is '1'B), the encoded data shall be represented as a complement of 2 by using s and val.

NOTE This encode rule is applied to the data structure when PDU is transferred, but need not specify the data structure on an actual memory in the data processing system in an actual device.



s: sign bit (0: + / 1: -)

val: positive value, if *s*=0;

"*s*+*val*" represents a negative value as a complement of 2, if *s*=1.

Figure 2 – Encode of Integer subtypes

When transferring the data of a multi-octets' data type, the octets shall be transmitted in order from the lower octet to the higher octet (see the example in Figure 3).

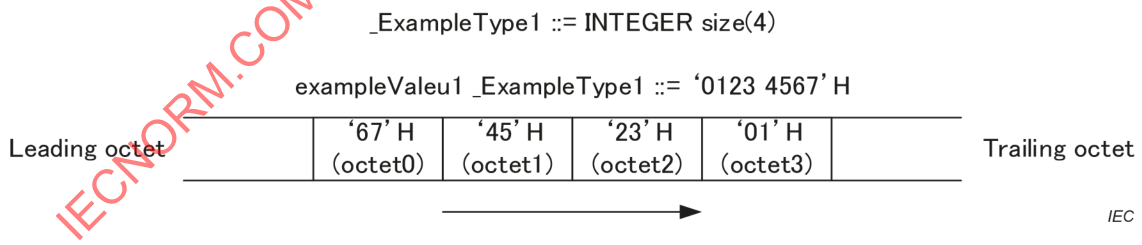


Figure 3 – Example of transfer of INTEGER value

In the encoded form of Unsigned8, Unsigned16, Unsigned32 and Unsigned64, the numeric value data (*val*) shall be set in the data area of its octet size as shown in Figure 4. In this case, the numeric value shall be set in order that the number in a lower digit is put into a lower bit in sequence, and 0 value shall be set to the unused upper bit.

When transferring data of a multi-octets' data type, the octets shall be transmitted in order from the lower octet to the higher octet.

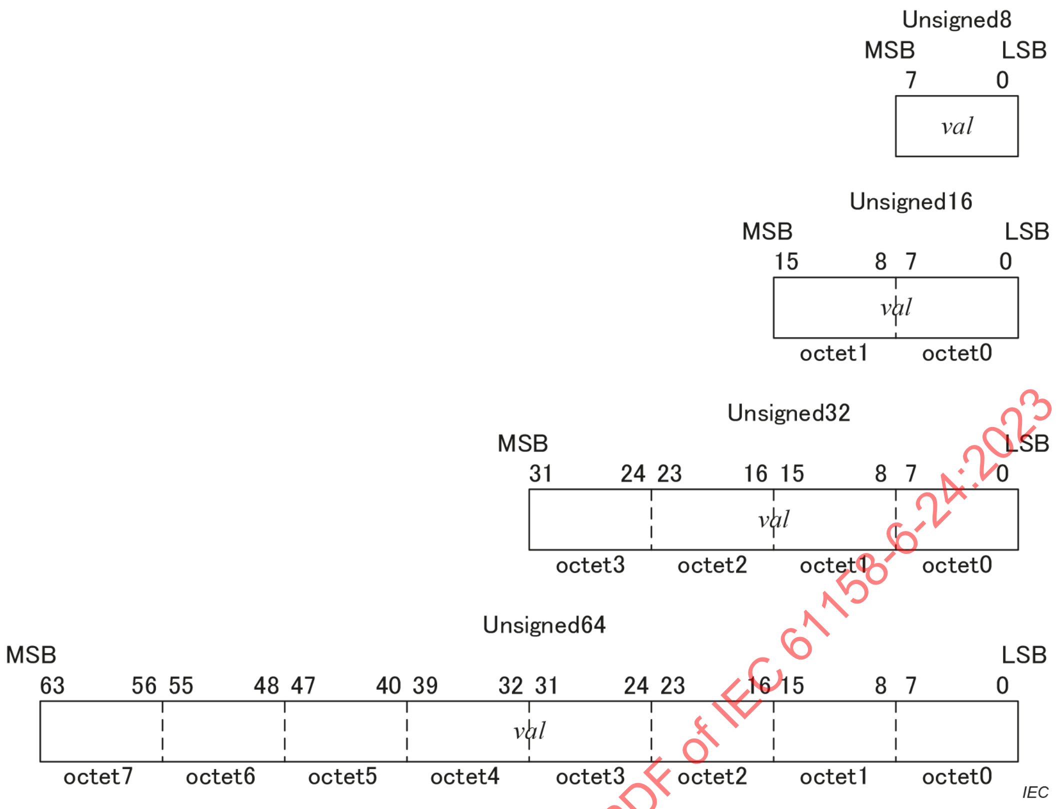


Figure 4 – Encode of Unsigned subtypes

5.2.2 REAL type and its subtypes

The Type 24 FAL does not directly define the data of REAL type within the abstract syntax, and uses $Float_{32}$ and $Float_{64}$ that are derivatives of REAL type. This encoding shall comply with ISO/IEC/IEEE 60559. The encoded formats of them are shown in Figure 5 and Figure 6.

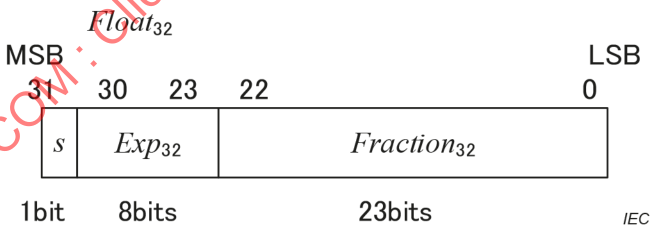


Figure 5 – $Float_{32}$ type encode

Data of $Float_{32}$ type shall be 4 octets' data as shown in Figure 4. The octets shall be transmitted in order from the lower octet to the higher octet. In this case, the value of the floating-point data of $Float_{32}$ type shall be calculated by using the following formula:

$$\mathrm{Float}_{32}=(-1)^{s}\times c_{32}\times 2^{q_{32}},$$

where

s is the sign bit (0:+ / 1:−);

$c_{32}=\left(1\times 2^{23}+Fraction_{32}\right)\times 2^{-23}=$

$1+\frac{b_{22}}{2}+\frac{b_{21}}{2^{2}}+\cdots+\frac{b_{0}}{2^{23}}$: mantissa;

$q_{32}=Exp_{32} - 127$: exponent (−127..127).

If both $Fraction_{32}$ and Exp_{32} equal 0, then $Float_{32}$ represents 0.

If $Fraction_{32}$ equals 0 and Exp_{32} equals 255, then $Float_{32}$ represents positive or negative infinity as the sign bit s.

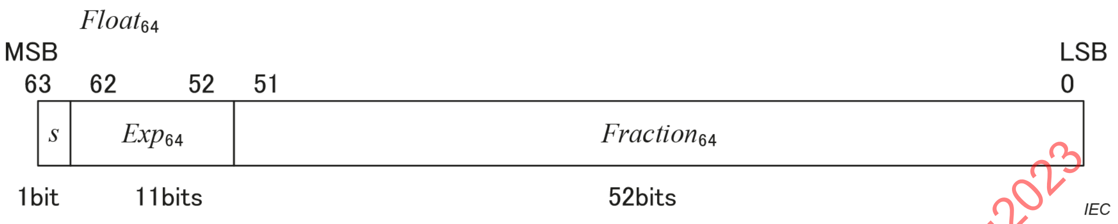


Figure 6 – $Float_{64}$ type encode

Data of $Float_{64}$ type shall be 8 octets' data as shown in Figure 6. The octets shall be transmitted in order from the lower octet to the higher octet. In this case, the value of the floating-point data of $Float_{64}$ type shall be calculated by using the following formula:

$$\mathrm{Float}_{64}=(-1)^{s}\times c_{64}\times 2^{q_{64}},$$

where

s is the sign bit (0:+ / 1:−);

$c_{64}=\left(1\times2^{52}+Fraction_{64}\right)\times2^{-52}$

$=1+\frac{b_{51}}{2}+\frac{b_{50}}{2^{2}}+\cdots+\frac{b_{0}}{2^{52}}$: mantissa;

$q_{64}=Exp_{64} - 1023$: exponent (−1 023..1 023).

If both $Fraction_{64}$ and Exp_{64} equal 0, then $Float_{64}$ represents 0.

If $Fraction_{64}$ equals 0 and Exp_{64} equals 2 047, then $Float_{64}$ represents positive or negative infinity as the sign bit s.

5.2.3 BIT STRING type

In BIT STRING type, the leading bit (0th bit) shall be corresponding to the LSB of the encoded data and the trailing bit shall be corresponding to the MSB according to the bit numbers attached to the named bits. However, the padding data shall be fulfilled by the octet boundary. In this case, an area can be reserved even for undefined bit; however, the value of the bit is not defined. Figure 7 shows an example of the bit field definition in the BIT STRING type.

For a BIT STRING data block of multi-octets'data size, the octets shall be transmitted in order from the octet with a lower bit.

NOTE The definition of "leading bit" and "trailing bit" is in accordance with ISO/IEC 8824-1:2008, 22.2.

```
_BitField1 ::=   BIT STRING{
                  bf1 (0),
                  bf2 (2),
                  bf3 (3),
                  bf4 (8)
                  }
```

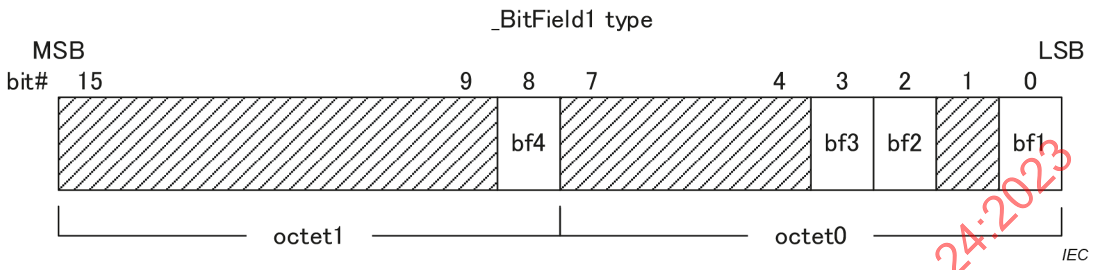


Figure 7 – Bit field definition example with named bits

A bit field is defined by combining SEQUENCE type and BIT STRING type. As shown in Figure 8, the boundary of each field shall be defined according to the specified bit size and filled from the lower bit.

To encode this bit field, an octet shall be filled from its lower bit according to the order defined by the field definition described between "{" and "}" in the SEQUENCE syntax to define a data area that is confined by the octet boundary. When undefined fraction bits remain, they shall be a reserve area and their values shall be undefined.

```
_BitField2 ::=   SEQUENCE{
                  cf1 BIT STRING size(1),
                  cf2 BIT STRING size(2),
                  cf3 BIT STRING size(1),
                  cf4 BIT STRING size(4),
                  cf5 BIT STRING size(1),
                  cf6 BIT STRING size(3)
                  }
```

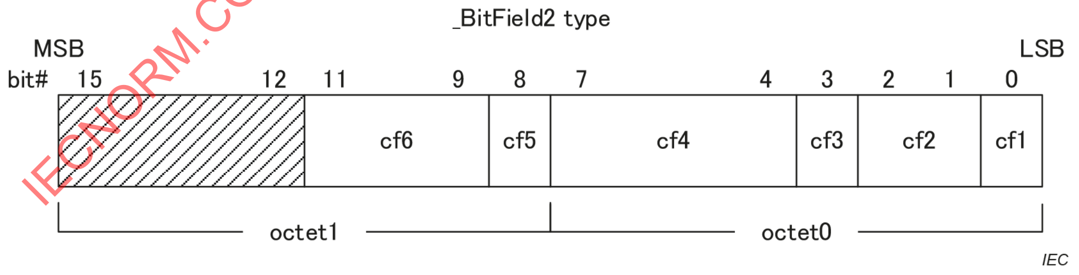


Figure 8 – Bit field definition example with field size

5.2.4 OCTET STRING type and IA5String type

Data of OCTET STRING type shall be transferred without converting the code in order from the most significant octet of the given string data.

Data of IA5String type shall be encoded into a data row (OCTET STRING) by converting the given String data into a 1-octet code, according to ISO/IEC 646, one character by one character, and then add one octet of null code ('00'H) to the last character code. The data shall be transferred in order from the first character code data.

5.2.5 NULL type

For data of NULL type, the data length is 0 and no value exists. Therefore, it shall not be encoded nor transferred.

5.2.6 Structure type and Array type

Among the elements of the Structure type, the SEQUENCE type and SEQUENCE OF type themselves shall not be encoding. Components included in these types shall be encoded and transferred according to each rule and the order of octet transfer, as shown in Figure 9.

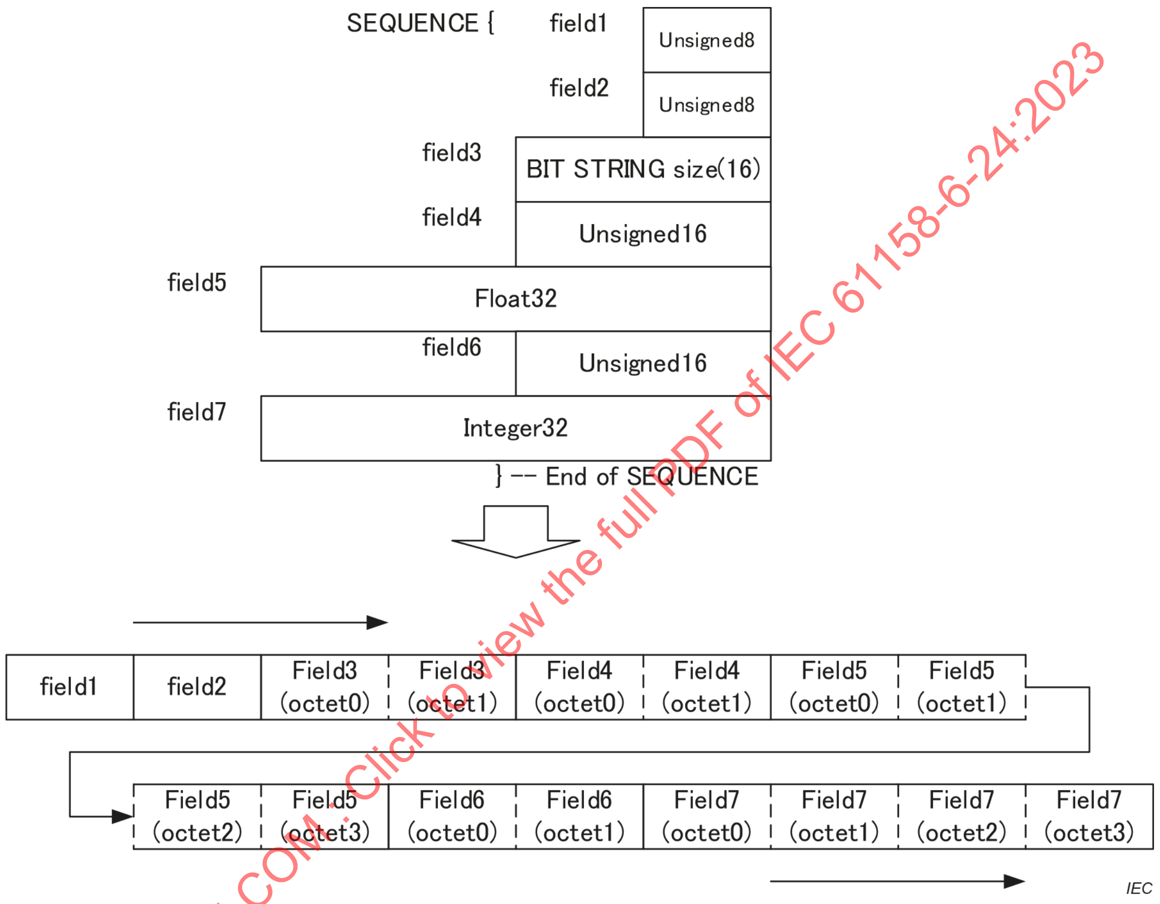


Figure 9 – SEQUENCE type encode

Also, the CHOICE type itself shall not be encoding. To edit communication data in any one of multiple alternatives of this type, the data should be selected according to the context of the communication AP and the agreement between the communication devices should be established. The method for this process depends on implementation and out of range of this document. The data shall be encoded according to the rule of the selected component.

6 Structure of FAL protocol state machine

In this clause and subsequent ones, FAL ASEs are characterized with protocol state machine (PM) models.

Although this type of fieldbus has its own structure of PMs, a mapping table is shown below for a clear understanding (see Table 4). It maps the structure for this type onto the typical four-sublayered one adopted by most of other types, which contains AP-Context PM, FSPM, ARPM, and DMPM.

Figure 10 shows the structure of PMs.

- There is a formally defined AP-Context State Machine (APC SM) for FAL user's initiation of a specific communication application.
- There is no formal definition of FSPM Machine just serving as an interface between FAL User and ARPM. Instead of that, two types of protocol machines are defined:
  - a set of Field Device Control machines (FDC PM) as master, slave, and monitor for FDC service users;
  - a pair of Message machines (MSG PM) as requester and responder for Message service users.
- Two different types of ARPM Machines are defined at the interface to the Data Link layer (DLL):
  - a set of ARPM machines for connection-oriented application relationships between classes of Master, Slave or Monitor of FDC Service;
  - a kind of ARPM machines for connection-less application relationships between classes of Requester and Responder of Messages Service.
- There is no formal definition of the DLL Mapping Protocol Machine (DMPM), unified into ARPM or APCSM instead. And DL services are used directly by ARPM or APCSM.

It is assumed that FDCPM functions in FDC ASE; and also MSGPM in MSG ASE; ARPM in AR ASE; and AP-CONTEXT SM in FSM ASE.

Table 4 – Mapping for Protocol State Machines

AP Type	ASE	Activated Class	Protocol Machine	Mapping to a typical structure for most of other types
C1 Master AP	FSM ASE	FieldbusSystemManager	APC SM	AP-Context-PM
	FDC ASE	Master	FDC PM-M	FSPM
	MSG ASE	Requester	MSG PM-RQ	FSPM
		Responder	MSG PM-RS	
	AR ASE	FDCMaster-AR	ARPM-FDCM	ARPM with DMPM
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a
Slave AP	FSM ASE	FieldbusSystemManager	APC SM	AP-Context-PM
	FDC ASE	Slave	FDC PM-S	FSPM
		Monitor (option)	FDC PM-MN	
	MSG ASE	Responder	MSG PM-RS	FSPM
	AR ASE	FDCSlave-AR	ARPM-FDCS	ARPM with DMPM
		FDCMonitor-AR	ARPM-FDCMN	
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a
C2 Master AP	FSM ASE	FieldbusSystemManager	APC SM	AP-Context-PM
	FDC ASE	Master(option)	FDC PM-M	FSPM
		Slave(option)	FDC PM-S	
		Monitor	FDC PM-MN	
	MSG ASE	Requester	MSG PM-RQ	FSPM
		Responder	MSG PM-RS	
	AR ASE	FDCMaster-AR(option)	ARPM-FDCM	ARPM with DMPM
		FDCSlave-AR(option)	ARPM-FDCS	

AP Type	ASE	Activated Class	Protocol Machine	Mapping to a typical structure for most of other types
		FDCMonitor-AR	ARPM-FDCMN	
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a

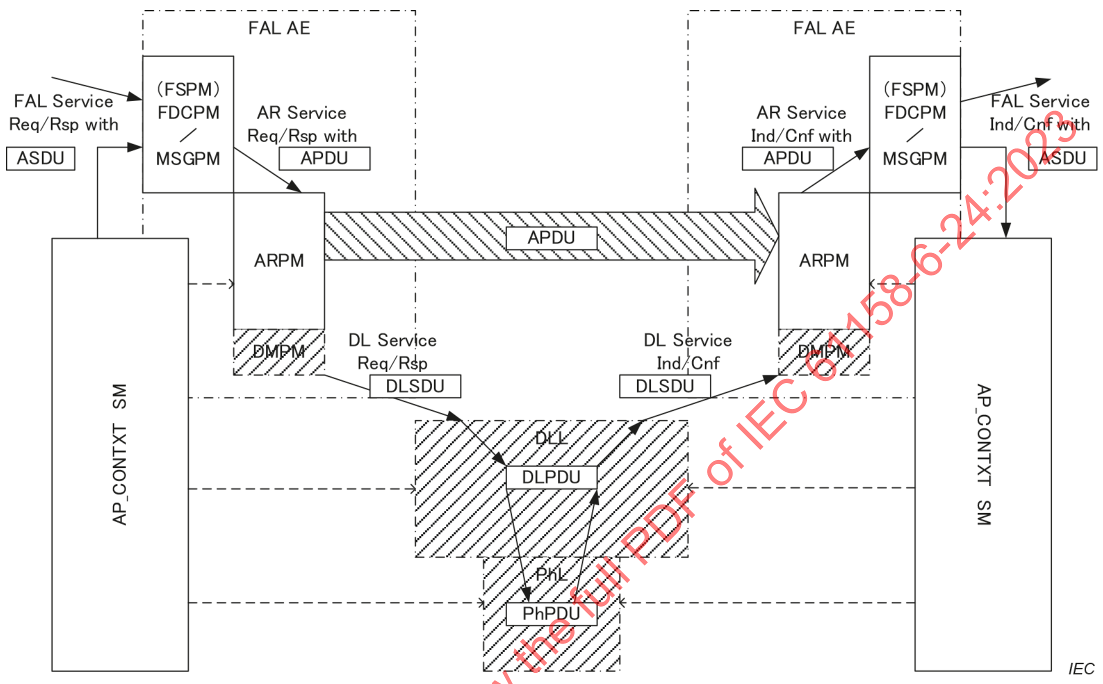


Figure 10 – Structure of FAL protocol state machines

7 AP-context state machine (APC SM)

7.1 Overview

An AP-context is a set of information and rules related to a communication system. It shall be created by instructions from an FAL user while the application process (AP) is invocated. For example, it contains selected communication parameters, based on the system configuration, an application field of the system, the own device profile, and difference of communication abilities between own device and peer ones.

In this ASE model, an FAL should retain the information mentioned above within attributes of the FieldbusSystemManager class according to user's instructions or configurations. The information should be able to characterize the AP types such as C1 Master, C2 Master, Slave and Monitor Slave. In addition, it should be able to configure other communication parameters and the device profile, such as a servo drive, an inverter drive or an I/O device, in order to establish the communication environment.

The APCSM shall control a series of state transitions, in which a device boots up, the APCSM shall get the AP-context data from the FAL user, and it shall initialize the FAL and the lower layer with the context. In consequence, it shall enable steady communication. The APCSM realizes the service that the FSM ASE provides.

The APCSM need not directly edit an APDU, nor realize a protocol to transfer PDUs. However, it indirectly acts as a trigger of PDU-exchange by using the other ASE service and the lower layer service during the FAL initialization process.

Figure 11 shows the APCSM statechart diagram.

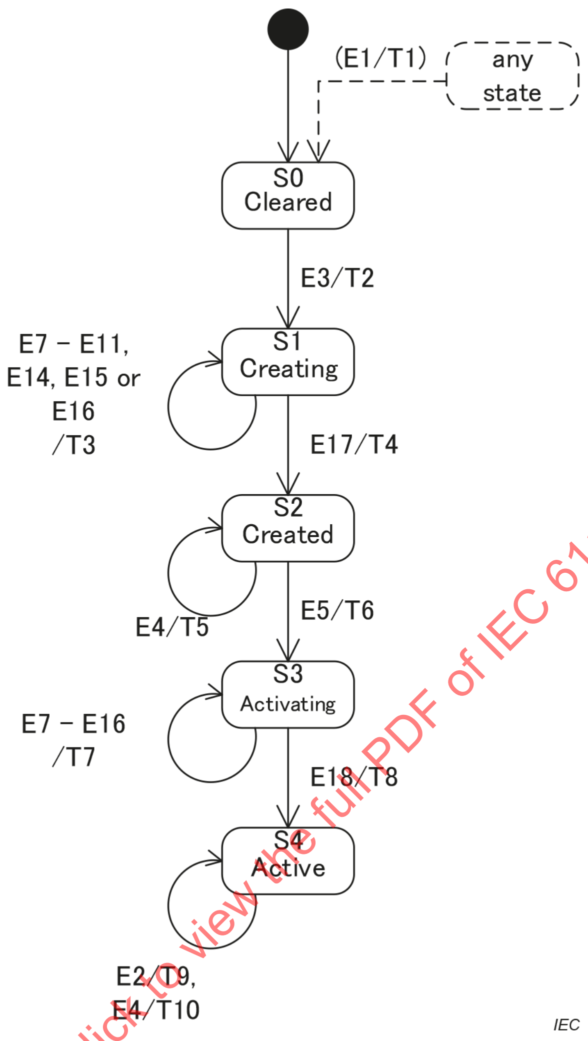


Figure 11 – Statechart diagram of APCSM

7.2 State descriptions

Table 5 describes each state of the APCSM.

Table 5 – State descriptions of APC SM

S#	State	Substate	Description
S0	Cleared	-	State when the FieldbusSystemManager Class of FSM ASE has just been instantiated as this SM All the entities in each layer and ASEs in the device are reset to the initial state, and AP-context is cleared.
S1	Creating		Transient state where AP-context or communication environment is generated, based on the input CONTEXT-DATA and the result of negotiation between the corresponding devices
S2	Created	-	State when AP-context has been generated
S3	Activating	-	Transient state when entities in each layer and ASEs are sequentially enabled
S4	Active	-	Steady state when all entities are activated and normal communication services are provided

7.3 Triggering events

Table 6 lists each trigger event of the APCSM.

Table 6 – Trigger event descriptions of APC SM

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FSM-Reset.req	FAL user		
E2	FSM-GetStatus.req	FAL user	INFO-ID	
E3	FSM-SetContext.req	FAL user	CONTEXT-DATA	
E4	FSM-GetContext.req	FAL user		
E5	FSM-Start.req	FAL user		
E6	<s>-Open.cnf	<s> ASE	ServiceStatus	
E7	DLM-SET-VALUE.cnf	DLM	Result	
E8	DLM-GET-VALUE.cnf	DLM	Result, Val	
E9	DLM-DELAY.ind	DLM	Delay_Time	
E10	DLM-DELAY.cnf	DLM	Delay_Time	
E11	DLM-SET-COMMODE.cnf	DLM	Result	
E12	DLM-START.ind	DLM	Com_Mode, Cycle_time, C2_stime, Max_Delay, TM_unit	
E13	DLM-START.cnf	DLM	Result	
E14	DLM_CLR-ERR.cnf	DLM	Result	
E15	Ph-SET-VALUE.cnf	PhL		
E16	Ph-GET-VALUE.cnf	PhL	Value	
E17	APC-Created	APCSM (S1 state)		Internal event
E18	APC-Activated	APCSM (S3 state)		Internal event

7.4 Action descriptions at state transitions

Detail specifications depending on its implementation are out of the scope of this document, for example such about initialization process, activate timing and various communication parameters in each layer and ASEs. Just outline of the related action is shown in Table 7.

Table 7 – Transitions of APC SM

T#	Source State	Event (arguments) [conditions] / action	Target State

T1	(any state)	E1:FSM-Reset.req
		/ Ph-RESET.req;
		DLM_RESET.req;
		EVM-Reset.req;
		for all AR ASE objects { AR-Reset.req;};
		for all FDC ASE objects { FDC-Reset.req};
		for all MSG ASE objects { MSG-Reset.req};
			S0:Cleared

T#	Source State	Event (arguments) [conditions] / action	Target State
T2	S0:Cleared	E3: FSM-SetContext.req (CONTEXT-DATA) / /*-- APCSM starts initializing PhL -- with Ph-SET-VALUE.req --*/; /*-- APCSM starts initializing DLL -- with DLM_SET_PAR.req -- and DLM_DELAY.req --*/; for all AR ASE objects { AR-Open.req;}; for all FDC ASE objects { FDC-Open.req}; for all MSG ASE objects { MSG-Open.req};	S1:Creating
T3	S1:Creating	E7, E8, E9, E10, E11, E14, E15, or E16 / /*-- APCSM keeps initializing PhL, DLL, and FAL --*/; /*-- If initializing procedures have finished, -- E17:APC-Created is issued. --*/;	S1:Creating
T4	S1:Creating	E17:APC-Created / FSM-SetContext.cnf;	S2:Created
T5	S2:Created	E4:FSM-GetContext.req / FSM-GetContext.cnf;	S2:Created

T6	S2:Created	E5:FSM-Start.req
		/ /*-- start activating PhL --*/;
		/*-- start activating DLL --*/;
		EVM-Enable.req;
		for all AR ASE objects { AR-Enable.req;};
		for all FDC ASE objects { FDC-Enable.req};
		for all MSG ASE objects { MSG-Enable.req};
			S3:Activating

T7	S3:Activating	E7, E8, E9, E10, E11, E12, E13, E14, E15, or E16 / /*-- APCSM keeps activating PhL, DLL, FAL --*/; /*-- If activating procedures have finished, -- E18:APC-Activated is issued --*/;	S3:Activating
T8	S3:Activating	E18: APC-Activated / FSM-Start.cnf;	S4:Active
T9	S4:Active	E2: FSM-GetStatus.req (INFO-ID) / /*-- APCSM reads appropriate status info for INFO-ID, -- using Ph-GET_VALUE.req, -- DL_GET_STATUS.req, -- DLM_GET_ERR.req -- or other service primitives --*/; FSM-GetStatus.cnf;	S4:Active
T10	S4:Active	FSM-GetContext.req / FSM-GetContext.cnf;	S4:Active

8 FAL service protocol machines (FSPM)

8.1 Overview

When the FSPM receives a service request primitive or a response primitive from an FAL user, then it shall edit an APDU from an SDU as the parameter of the primitives, and then request the ARPM to transmit. It shall also take out an SDU from the APDU received by the ARPM to deliver as an indication primitive or a conform primitive to the FAL user.

Two types of application services (FDC ASE and MSG ASE) can be provided to the FAL user in the Type 24 FAL (refer to IEC 61158-5-24), and the FSPM consists of two types of PM (FDC PM and MSG PM) corresponding to the ASEs.

8.2 Field Device Control Protocol Machine (FDC PM)

8.2.1 Protocol overview

The FDC PM is a protocol state machine (PM) that realizes the services provided by the FDC ASE. The protocols are categorized as shown in Table 8.

Table 8 – FDC protocol mode

Transmission mode (by DLL)	Communication state	Communication command type	Description
Cyclic	Synchronous communication (Sync)	Synchronous communication type command (Sync command)	The FDC ASE, both the master and the slave, shall notify firing of an event to the user in each communication cycle with FDC-ComCycle.ind. The user of the FDC master can update a next command to the slave AP in each communication cycle to request to transfer it without waiting the process completion response (FDC-Command.cnf) to the previous command. The FDC slave shall receive a command from the master in each communication cycle and pass it to the user of the FDC slave with FDC-Command.ind. The user should process the command, then return the response (FDC-Command.rsp) within the one communication cycle.
		Asynchronous communication type command (Async command)	The FDC ASE, both the master and the slave, shall notify firing of an event to the user in each communication cycle with FDC-ComCycle.ind. The user of the FDC master should request to transfer a command to the slave AP in each communication cycle while waiting the process completion response to the previous command. The contents of the command should be maintained until the user received the process completion response. The FDC slave shall receive a command from the master AP in each communication cycle and pass it to the user. The user should return a corresponding response (FDC-Command.rsp) with a command progress status (cmdRdy or busy) in each communication cycle and should not accept any new command until the user completes the processing command.

Transmission mode (by DLL)	Communication state	Communication command type	Description
	Asynchronous communication (Async)	Async command	The FDC ASE, both the master and the slave, shall notify firing of an event to the user in each communication cycle with FDC-ComCycle.ind. Once the user of the FDC master requests to transfer a command to the slave AP, the user should wait the corresponding process completion response to the previous command. Until then, the user should not request to transfer any updated command. The FDC slave receives a command from the master AP in each communication cycle. When the contents of the command are updated, the FDC slave shall pass it to the user. The user should return a corresponding response (FDC-Command.rsp) with a command progress status (cmdRdy or busy) in each communication cycle and shall not accept any new command until the user completes the processing command.
Event-driven	Async	Async command	The FDC ASE, both the master and the slave, shall not notify any event to the user in a regular cycle because no communication cycle is generated. Once the user of FDC master requests to transfer a command, the user should wait the corresponding process completion response for the previous command and should not transfer any updated command. The FDC slave shall pass the user the command received from the master AP. The user should not return any response nor accept any new command until processing the received command is completed or the pre-defined processing time elapses.

8.2.2 Cyclic communication mode

8.2.2.1 Cyclic communication common specifications

In the cyclic communication mode, an event shall be generated in a constant period, referred as the communication cycle. The communication process provided by the FDC ASE shall be executed repeatedly with this event. The communication cycle shall originate in the cycle of an integral multiple of the transmission cycle managed by the data link layer.

The transmission cycle should be a constant cycle event generated with the periodic running counter synchronized with all devices.

The communication cycle of connection type shall start when the connection has been established between the master and the slave. Therefore, before the connection is established, even in the cyclic communication mode, the communication process cannot be synchronized with the communication cycle but the primary cycle or the transmission cycle, and only communication by using certain asynchronous type commands can be executed.

The communication cycle of connection-less type shall start when the C1 master AP has issued the cyclic communication start request.

The protocol for the establishment and release of the connection shall be also provided as the FDC ASE services.

The communication cycle in the FDC ASE can have some effects on even the user process, such as C1 master AP or slave AP, see Figure 12, Figure 13.

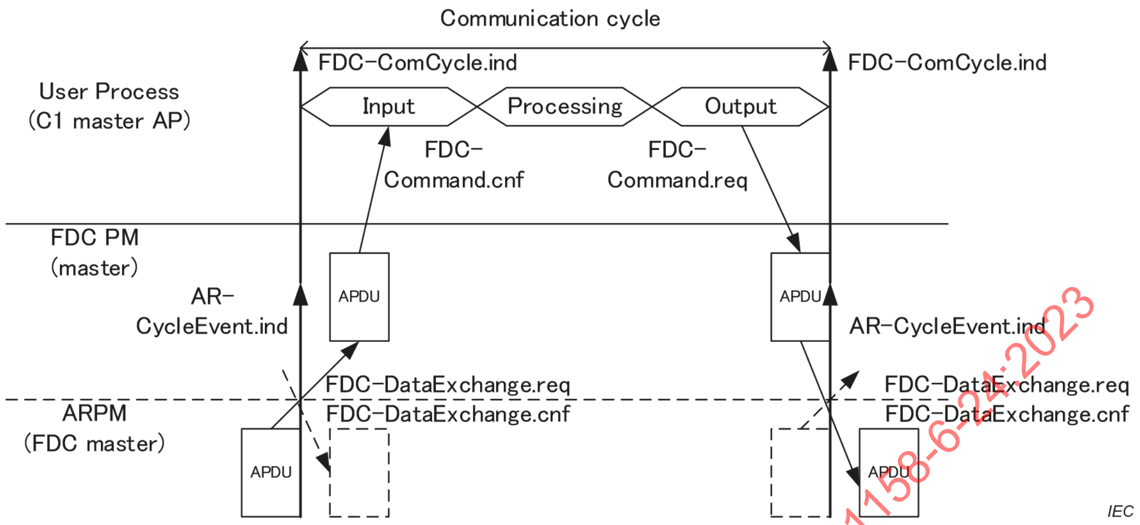


Figure 12 – Example communication cycle of FDC master AP

For example, in the C1 master, the command output timing, the response input timing and the data processing timing is handled most efficiently in the case when they are processed with the cycle event (FDC-ComCycle.ind) as indicated in Figure 12.

NOTE The Figure 11 and Figure 12 are shown just as informative examples.

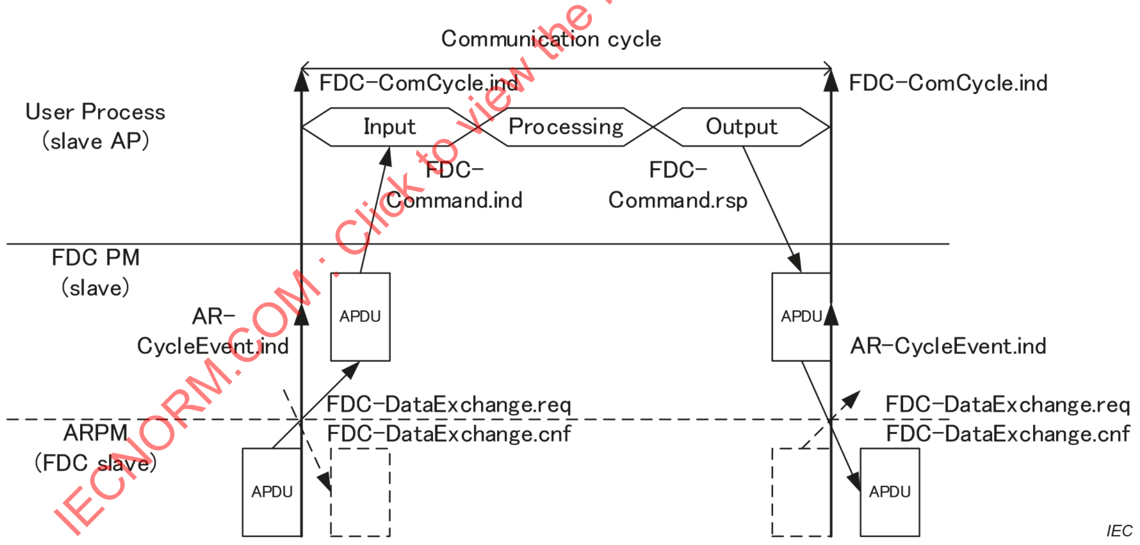


Figure 13 – Example communication cycle of FDC slave AP

In the same way for the slave AP, the command input timing, the response output timing and the data processing timing is handled most efficiently in the case when they are processed as indicated in Figure 13.

8.2.2.2 Synchronous communication state (SyncConnected)

The word "synchronous" within the "synchronous communication state" and "synchronous communication type command" means that also the user processes issuing the commands in the states should be initiated or executed cyclically, as well as the FDC ASE communication process synchronizes with the communication cycle and is executed periodically.

The master AP should request to transfer a command and the slave AP should request to transfer a response respectively once in each communication cycle when the FDCPM is in the synchronous communication state. In this case, the master AP and the slave AP should watch over the status of the synchronization activity each other by using a counter to watch or a _WDT (Watch Dog Timer: WDT) field on each SDU.

When the master AP transfers synchronous type commands continuously for two or more times, it can request to do them in each communication cycle one after another without waiting to receive the corresponding response. The slave AP should complete processing the synchronous command within the communication cycle in which the command is received and transmit the response. The master AP and the slave AP can exchange this type of command and response with the WDT counter or an appropriate counter to acknowledge that they are synchronized with each other.

Figure 14 shows the timing chart for the synchronous command communication.

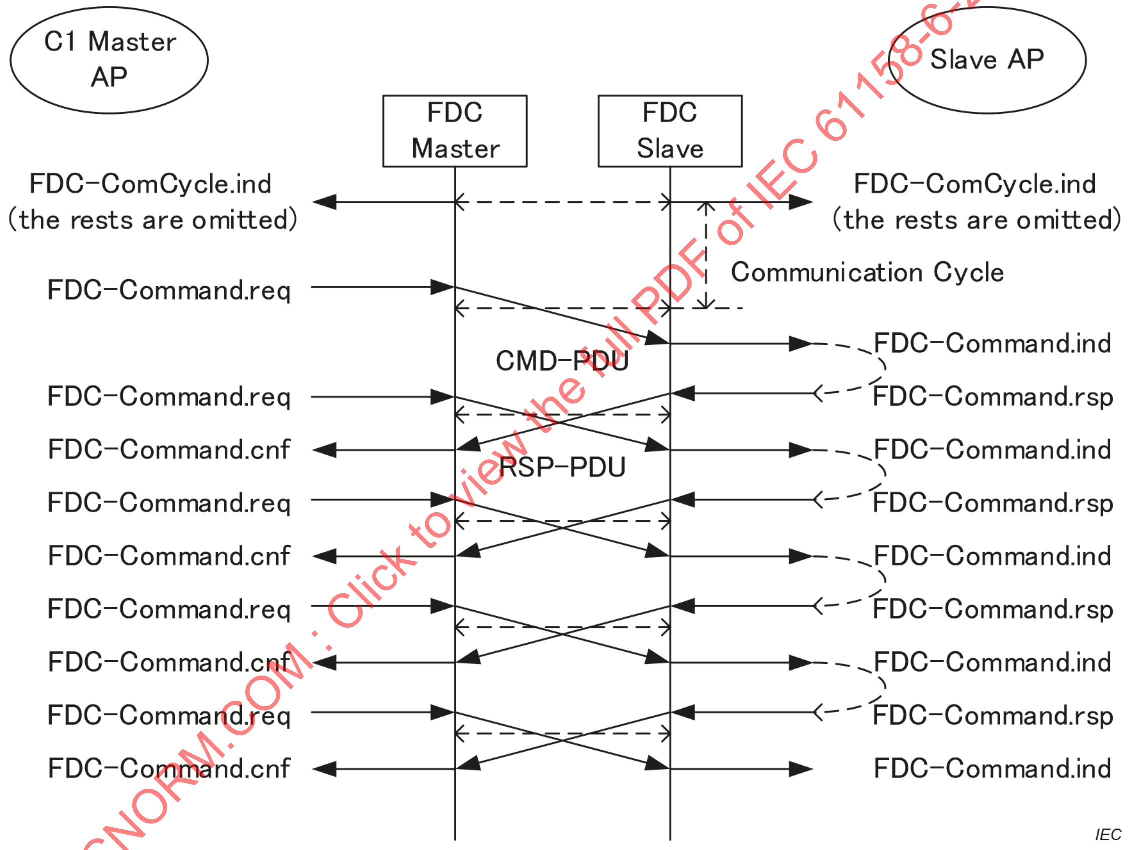


Figure 14 – Synchronous command communication in sync state

When the master AP transfers an asynchronous type command, the master AP should request to transfer the next command after it has confirmed the corresponding response to the previous command indicating the process completes by a cmdRdy-bit = 1 or "command ready", or busy = 0.

However, even in that case, the master AP should continuously request to transfer the command with the same contents in every communication cycle. And the slave AP should request to transfer the response with a cmdRdy-bit = 0 or "command busy", so that each watchdog viewing of the synchronized status through the WDT or an appropriate counter succeeds.

NOTE The names such as "synchronous type command" and "asynchronous type command" represent whether the user process is executed and completed while being synchronized with the communication cycle or not. On the other hand, from the aspect of the transaction management for the command and the response, it can be said that the synchronous type command processes asynchronous transactions, and the asynchronous type command processes synchronous transactions.

Figure 15 shows the timing chart for the asynchronous command communication in the synchronous state.

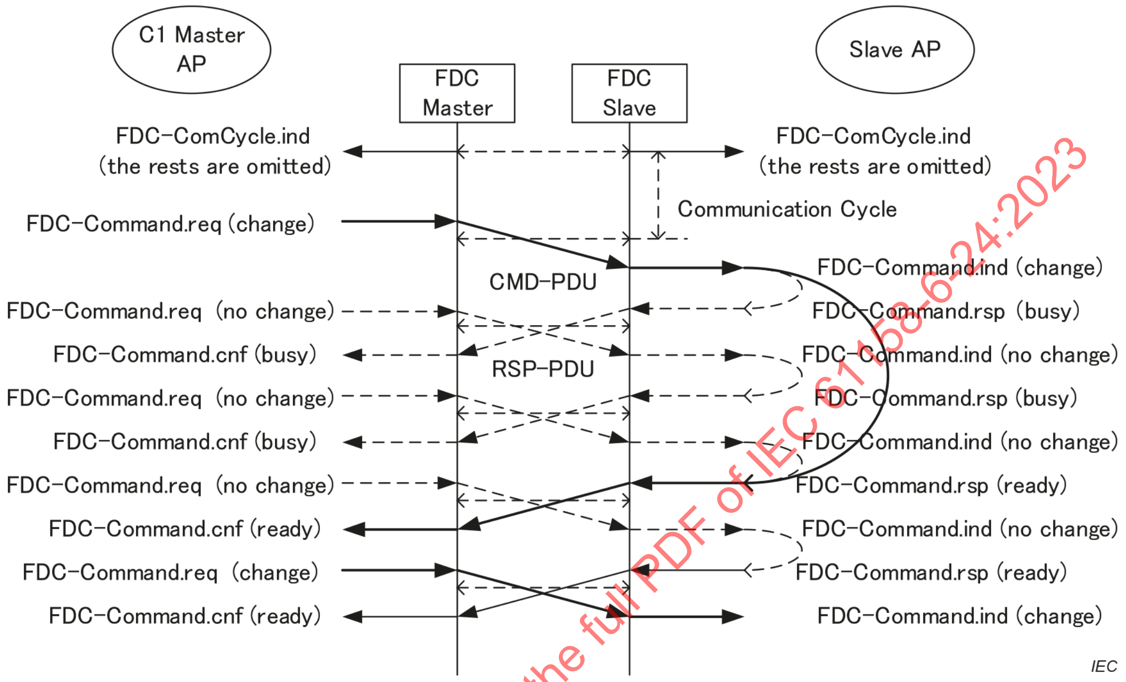


Figure 15 – Asynchronous command communication in sync state

8.2.2.3 Asynchronous communication state (AsyncConnected)

Even under the asynchronous transmission state, the communication process of the FDC ASE shall be periodically initiated synchronous with the communication cycle, because it is in the cyclic communication mode. In this case, the master AP and the slave AP should not use a _WDT (Watch Dog Timer: WDT) field on each SDU to watch over the synchronization activity.

Therefore, constant periodic event with FDC-ComCycle.ind can be notified to the user process in each communication cycle. However, the user process need not always be operated synchronous with the event.

In this state, the FDC ASE shall provide the user process with only asynchronous type commands. Therefore, the master AP can transfer a new command only after it confirms the corresponding process completion response to the transferring command.

Figure 16 shows the timing chart in the asynchronous communication state.

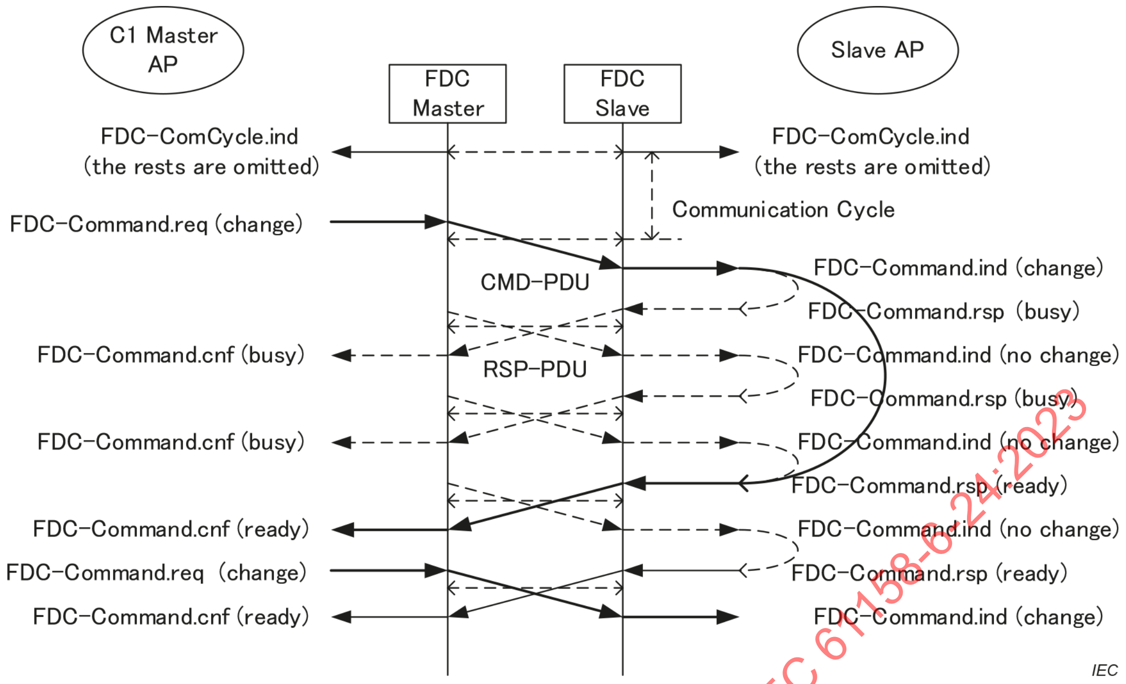


Figure 16 – Asynchronous command communication in async state

8.2.3 Event driven communication mode

In the event driven communication mode, the FDC ASE can have no communication cycle and the communication process shall not be executed cyclically. The network clock cannot function either.

In this mode, the FDC ASE provides no service related to the synchronous type command and the user process can only use asynchronous type commands. And the master AP can request to transfer a command non-periodically whenever the preceding command transaction has already completed.

Figure 17 shows the timing chart in the event driven mode communication.

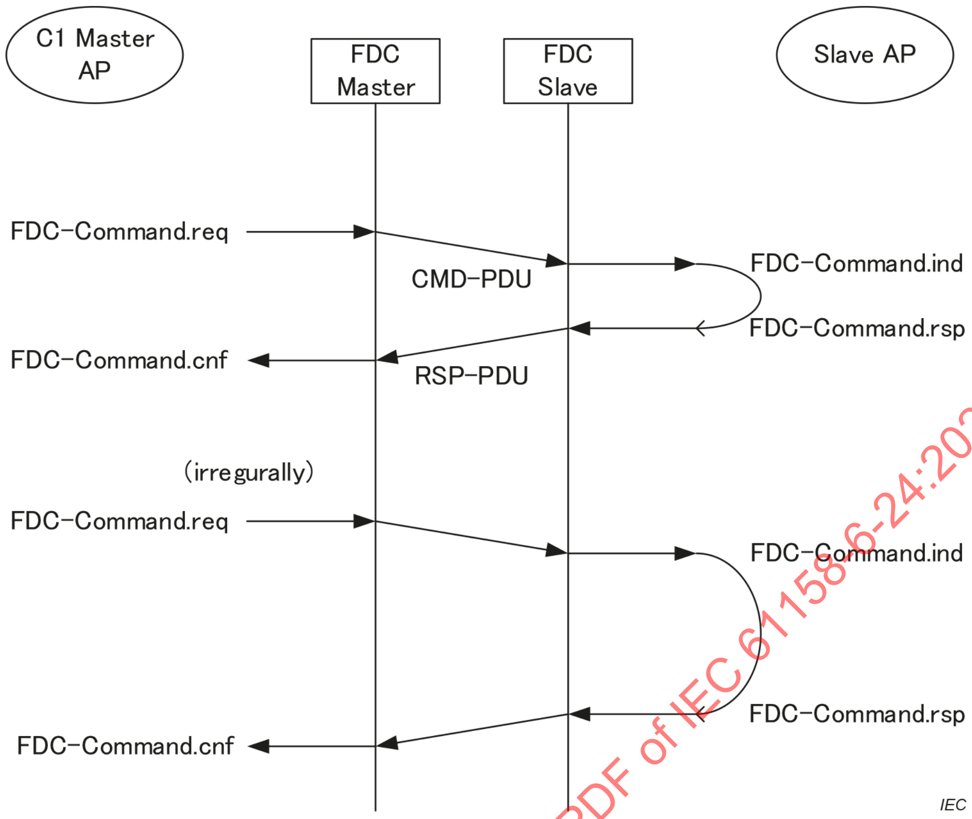


Figure 17 – Event-driven communication

8.2.4 Master Protocol Machine (FDCPM-M)

8.2.4.1 State descriptions

Figure 18 shows the FDCPM-M statechart diagram, and Table 9 describes each state of the FDCPM-M.

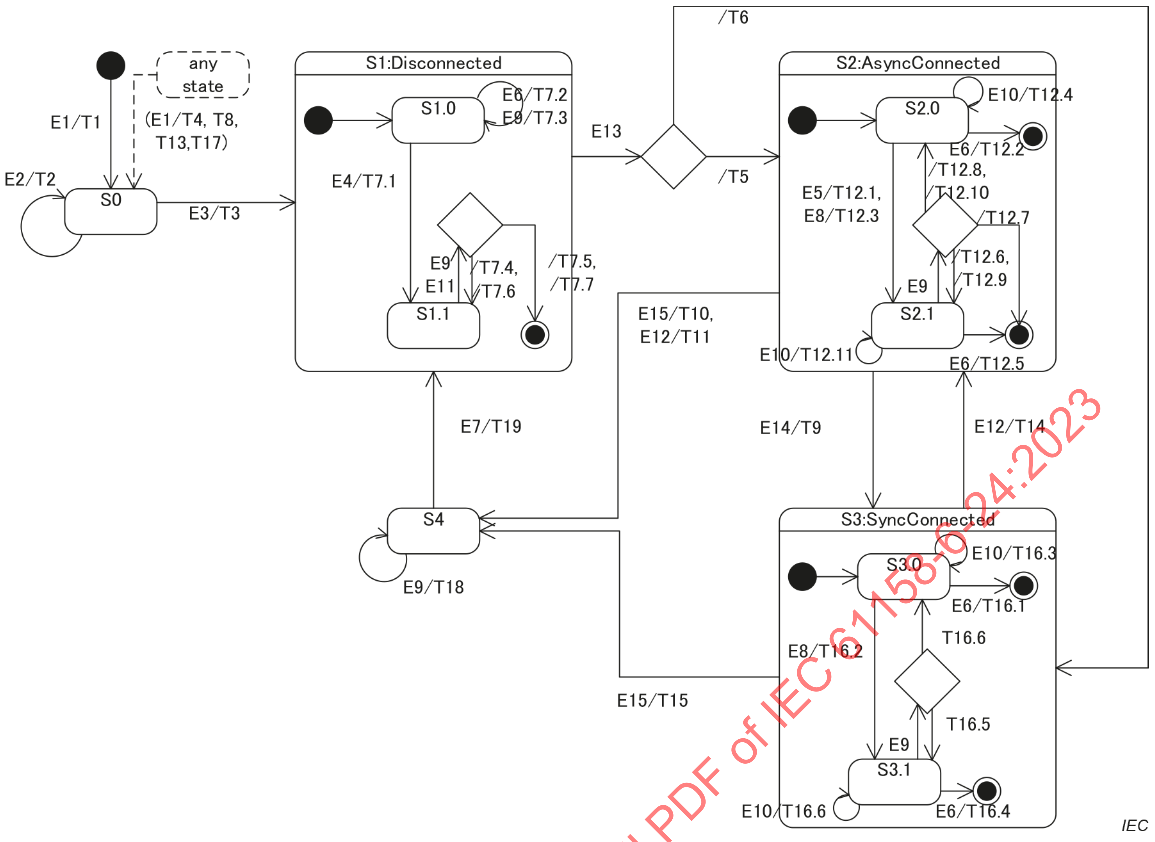


Figure 18 – Statechart diagram of FDCPM-M

Table 9 – State descriptions of FDCPM-M

S#	State	Substate	Description
S0	Disabled	-	State when the Master Class of FDC ASE (FDC Master) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from FSM ASE. --- Entry/ Initial values are set in Attributes of the Master object.
S1	Disconnected	-	State when the connection is released and the PM is waiting Connect.request from a FAL user In the lower layer, a communication has been started. Only the connection control commands (CONNECT, DISCONNECT) and NOP command is allowed to be transferred in this state. Even if the DL layer is running in the cyclic communication mode, the communication cycle has not been notified in this state because no connection is established.
S1.0		Idle	Substate when the PM is waiting a request to transfer a CMD-PDU from a FAL user
S1.1		WaitCMDComplete	Substate when the PM is waiting a command completion response from the correspondent peer Slave AP

S#	State	Substate	Description
S2	AsyncConnected	-	A normal operation state to control a Slave AP or a function of a field device through the peer FDC Slave by using asynchronous type command A connection with the FDC Slave is established and a communication cycle event is notified by AR ASE. That event is transferred as FDC-ComCycle.ind to the C1 master AP. Any asynchronous type command can be transferred in this state. No synchronous type command is allowed to be transferred.
S2.0		Idle	Substate when the PM is waiting a request to transfer a CMD-PDU from a FAL user
S2.1		WaitCMDComplete	Substate when the PM is waiting a command completion response from the correspondent peer Slave AP
S3	SyncConnected	-	The most time-critical operation state to control a Slave AP or a function of a field device through the peer FDC slave by using both synchronous and asynchronous commands A connection with the FDC Slave has been established and a communication cycle event is notified by AR ASE. That event is transferred as FDC-ComCycle.ind to the C1 master AP. The WDT counter or an appropriate counter on every CMD-PDUs shall function to watch over the status of the synchronization activity of the C1 master AP. And so as to the RWDT counter or an appropriate counter on RSP-PDUs for the Slave AP.
S3.0		Idle	Substate when the PM is waiting a request to transfer a CMD-PDU from a FAL user
S3.1		WaitCMDComplete	State when the connection changes to be released because communication errors continuously occur or the FAL user demands it. The PM waits in order to prevent the system malfunctioning as a kind of interlocking mechanism, and then recovers to S1 state when receiving a resume request from the user (ResumeCycle.req).
S4	Disconnecting	-	State when the connection changes to be released because communication errors continuously occur or the FAL user demands it. The PM waits in order to prevent the system malfunctioning as a kind of interlocking mechanism, and then recovers to S1 state when receiving a resume request from the user (ResumeCycle.req).

8.2.4.2 Triggering events

Table 10 lists each trigger event of the FDCPM-M.

Table 10 – Trigger event descriptions of FDCPM-M

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FDC-Reset.req	FSM ASE		
E2	FDC-Open.req	FSM ASE	AREPID	
E3	FDC-Enable.req	FSM ASE	TransmissionMode	
E4	FDC-Connect.req	FAL user (C1masterAP)	Update, CONNECT-CMD-SDU	
E5	FDC-SyncSet.req	FAL user (C1masterAP)	Updata, SYNC_SET-CMD-SDU	
E6	FDC-Disconnect.req	FAL user (C1masterAP)	DISCONNECT-CMD-SDU	
E7	FDC-ResumeCycle.req	FAL user (C1masterAP)		
E8	FDC-Command.req	FAL user (C1masterAP)	Update, CMD-SDU	
E9	FDC-DataExchange.req	AR ASE (FDCM-AR)	RSP-SDU	
E10	AR-CycleEvent.ind	AR ASE (FDCM-AR)	NetworkClock	
E11	AR-SendCommand.cnf	AR ASE (FDCM-AR)	ServiceStatus, RSP-SDU	
E12	Error detected	This object		See 8.2.7 and notes of Table 11 WDT failures are detected twice serially under SyncConnect state. Alternatively, failures are detected continuously under AsyncConnect state.
E13	EventConnectRSP	This object (submachine)	status Unsigned16 rsdu _CONNECT-RSP-PDU	Notification from submachine Connection-less type
E14	EventSyncSetRSP	This object (submachine)	status Unsigned16 rsdu _SYNC_SET-RSP-PDU	Notification from submachine
E15	EventDisconnectCMD	This object (submachine)	csdu _DISCONNECT-CMD-PDU	Notification from submachine

8.2.4.3 Action descriptions at state transitions

Table 11 describes state transitions of the main SM of FDCPM-M. Moreover, Table 12 describes state transitions of the submachine of FDCM-M.

Table 11 – Transitions of main SM of FDCPM-M

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC master --*/;	S0:Disabled
T2	S0:Disabled	E2:FDC-Open.req / /*-- Initializing the FDC master --*/; [a]	S0:Disabled
T3	S0:Disabled	E3:FDC-Enable.req (communicationMode) /* Notifying the DLL's communication mode, initialized into whether the cyclic mode or the event-driven mode. */ / TransMode = communicationMode;	S1:Disconnected
T4	S1:Disconnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Disabled

T5	S1:Disconnected	E13:EventConnectRSP (status, rsdu) [rsdu.rspBody.syncmode != 1]
		/ AR-StartComCycle.req;
		ProtocolVersion = rsdu.rspBody.ver;
		SyncMode = 0;
		DTMode = rsdu.rspBody.dtmode;
		SubCMDMode = rsdu.rspBody.subcmd;
		ComTime =rsdu.rspBody.com_time;
		DevProfileType = rsdu.rspBody.profile_type;
			S2:AsyncConnected

T6	S1:Disconnected	E13:EventConnectRSP (status, rsdu) [rsdu.rspBody.syncmode == 1]
		/ AR-StartComCycle.req;
		ProtocolVersion = rsdu.rspBody.ver;
		SyncMode = 1;
		DTMode = rsdu.rspBody.dtmode;
		SubCMDMode = rsdu.rspBody.subcmd;
		ComTime = rsdu.rspBody.com_time;
		DevProfileType = rsdu.rspBody.profile_type;
			S3:SyncConnected

T7	S1:Disconnected	The other events / /*-- state-transition in the submachine --*/;	S1:Disconnected
T8	S2:AsyncConnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Disabled

T9	S2:AsyncConnected	E14:EventSyncSetRSP(status, rsdu)
		/ WDT.LastMN = 0;
		WDT.LastSN = 0;
		RWDT.LastRMN = 0;
		RWDT.LastRSN = 0;
			S3:SyncConnected

T#	Source State	Event (arguments) [conditions] / action	Target State
T10	S2:AsyncConnected	E15:EventDisconnectCMD (csdu) / /*-- no-operation --*/;	S4:Disconnecting
T11	S2:AsyncConnected	E12: Errors occur continuously [a, b,c] //*-- no-operation --*/;	S4:Disconnecting
T12	S2:AsyncConnected	The other events / /*-- state-transition in the submachine --*/;	S2:AsyncConnected
T13	S3:SyncConnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC master --*/	S1:Disabled
T14	S3:SyncConnected	E12: Errors occur serially twice. [b] //*-- no-operation --*/;	S2:AsyncConnected
T15	S3:SyncConnected	E15:EventDisconnectCMD(csdu) / /*-- no-operation --*/;	S4:Disconnecting
T16	S3:SyncConnected	The other events / /*-- state-transition in the submachine --*/;	S3:SyncConnected
T17	S4:Disconnecting	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC master --*/	S1:Disabled

T18	S4:Disconnecting	E9:FDC-DataExchange.req (rsdu)
		/ csdu.cmd = disconnect;
		FDC-DataExchange.cnf (csdu);
			S4:Disconnecting

T19	S4:Disconnecting	E7:FDC-ResumeCycle.req / /*-- Clearing internal information related to the communication cycle and alarm information --*/;	S2:Disconnected

[a] The detailed process depends on the system implementation.

[b] Implementers need some mechanisms to count or manage errors.

[c] The threshold of error duration count is an implementation matter.

Table 12 – Transitions of submachine of FDCPM-M

T#	Source State	Event (arguments) [conditions] / action	Target state

T7.1	S1.0:Idle	E4:FDC-Connect.req (csdu)
		/ LastCMD-SDU = csdu;
		If (TransMode == EventDriven)
		AR-SendCommand.req (csdu);
		else /*-- no-operation --*/;
			S1.1: WaitCMDComplete

T7.2	S1.0:Idle	E6:FDC-Disconnect.req (csdu)
		/ LastCMD-SDU = csdu;
		If (TransMode == EventDriven)
		AR-SendCommand.req (csdu);
		else /*-- no-operation --*/;
			S1.0: Idle

T7.3	S1.0:Idle	E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-DataExchange.cnf (csdu);
			S1.0:Idle

T7.4	S1.1:
	WaitCMDComplet e
		E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && ((rsdu.rcmd != LastCMD-SDU.cmd) || (status.cmdRdy != 1))]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-DataExchange.cnf (csdu);
			S1.1: WaitCMDComplete

T7.5	S1.1:
	WaitCMDComplet e
		E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU.cmd == connect)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-DataExchange.cnf (csdu);
		FDC-Connect.cnf (status, rsdu);
		/*-- signal E13:EventConnectRSP to the main machine --*/;
			Exit to the main machine

T7.6	S1.1:
	WaitCMDComplet e
		E11:AR-SendCommand.cnf (status, rsdu) [/*-- status has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		AR-SendCommand.req (csdu);
		/* Comment: TransMode is EventDriven. */
			S1.1: WaitCMDComplete

T7.7	S1.1:
	WaitCMDComplet e
		E11:AR-SendCommand.cnf (status, rsdu) [/*-- status has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU.cmd == connect)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-Connect.cnf (status, rsdu);
		/*-- signal E13:EventConnectRSP to the main machine --*/;
			Exit to the main machine

T#	Source State	Event (arguments) [conditions] / action	Target state
T12.1	S2.0:Idle	E5:FDC-SyncSet.req (csdu) / LastCMD-SDU = csdu;	S2.1: WaitCMDComplete

T12.2	S2.0:Idle	E6:FDC-Disconnect.req (csdu)
		/ LastCMD-SDU = csdu;
		If (TransMode == EventDriven)
		AR-SendCommand.req (csdu);
		else /*-- no-operation --*/;
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			Exit to the main machine

T12.3	S2.0:Idle	E8:FDC-Command.req (csdu)
		/ LastCMD-SDU = csdu;
		If (TransMode == EventDriven)
		AR-SendCommand.req (csdu);
		else /*-- no-operation --*/;
			S2.1: WaitCMDComplete

T12.4	S2.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S2.0: Idle

T12.5	S2.1:
	WaitCMDComplet e
		E6:FDC-Disconnect.req (csdu)
		/ LastCMD-SDU = csdu;
		If (TransMode == EventDriven)
		AR-SendCommand.req (csdu);
		else /*-- no-operation --*/;
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			Exit to the main machine

T12.6	S2.1:
	WaitCMDComplet e
		E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-DataExchange.cnf (csdu);
			S2.1: WaitCMDComplete

T12.7	S2.1:
	WaitCMDComplet e
		E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU.cmd == sync_set)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-DataExchange.cnf (csdu);
		FDC-SyncSet.cnf(+);
		/*-- signal E14:EventSyncSetRSP to the main machine --*/;
			Exit to the main machine

T12.8	S2.1:
	WaitCMDComplet e
		E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1) && (LastCMD-SDU != sync_set)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		FDC-DataExchange.cnf (csdu);
		FDC-Commnand.cnf (+);
			S2.0:Idle

T#	Source State	Event (arguments) [conditions] / action	Target state

T12.9	S2.1: WaitCMDComplete	E11:AR-SendCommand.cnf (status, rsdu) [/*-- status has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)]
		/ LastRSP-SDU = rsdu;
		csdu = LastCMD-SDU;
		If (TransMode == EventDriven)
		AR-SendCommand.req (csdu);
		else /*-- no-operation --*/;
			S2.1: WaitCMDComplete

T12.10	S2.1: WaitCMDComplete	E11:AR-SendCommand.cnf (status, rsdu) [/*-- status has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy == 1)]
		/ LastRSP-SDU = rsdu;
		FDC-Command.cnf (+);
			S2.0:Idle

T12.11	S2.1: WaitCMDComplete	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S2.1: WaitCMDComplete
T16.1	S3.0:Idle	E6:FDC-Disconnect.req (csdu) / LastCMD-SDU = csdu; /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit to the main machine
T16.2	S3.0:Idle	E8:FDC-Command.req (csdu) / LastCMD-SDU = csdu;	S3.1: WaitCMDComplete
T16.3	S3.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S3.0: Idle
T16.4	S3.1: WaitCMDComplete	E6:FDC-Disconnect.req (csdu) / LastCMD-SDU = csdu; /*-- signal E15:EventDisconnectCMD to the main machine --*/;	Exit to the main machine

T16.5	S3.1: WaitCMDComplete	E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy != 1)]
		/ LastRSP-SDU = rsdu;
		if (rsdu.rwdt.rsn != LastRSN)
		{/*-- signal E12 watchdog counter error to the main machine--*/;}
		else
		{LastRSN = rsdu.rwdt.rsn;
		csdu = LastCMD-SDU;
		LastSN = LastRSN;
		csdu.wdt.mn = LastMN;
		csdu.wdt.sn = LastSN;
		FDC-DataExchange.cnf (csdu);}
			S3.1: WaitCMDComplete
			or
			Exit to the main machine, when E12 has been issued.

T#	Source State	Event (arguments) [conditions] / action	Target state

T16.6	S3.1:
	WaitCMDComplete
		E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == CMD-SDU.cmd) && (status.cmdRdy == 1)]
		/ LastRSP-SDU = rsdu;
		if (rsdu.rwdt.rsn != LastRSN)
		{/*-- signal E12 watchdog counter error to the main machine --*/;}
		else
		{LastRSN = rsdu.rwdt.rsn;
		csdu = LastCMD-SDU;
		LastSN = LastRSN;
		csdu.wdt.mn = LastMN;
		csdu.wdt.sn = LastSN;
		FDC-DataExchange.cnf (csdu);
		FDC-Command.cnf (status, rsdu);}
			S3.0:Idle
			or
			Exit to the main machine, when E12 has been issued.

T16.7	S3.1:
	WaitCMDComplete
		E10: AR-CycleEvent.ind
		/ FDC-ComCycle.ind
		LastMN++;
		LastRSN++;
			S3.1:
			WaitCMDComplete

8.2.5 Slave Protocol Machine (FDCPM-S)

8.2.5.1 State descriptions

Figure 19 shows the FDCPM-S statechart diagram, and Table 13 describes each state of the FDCPM-S.

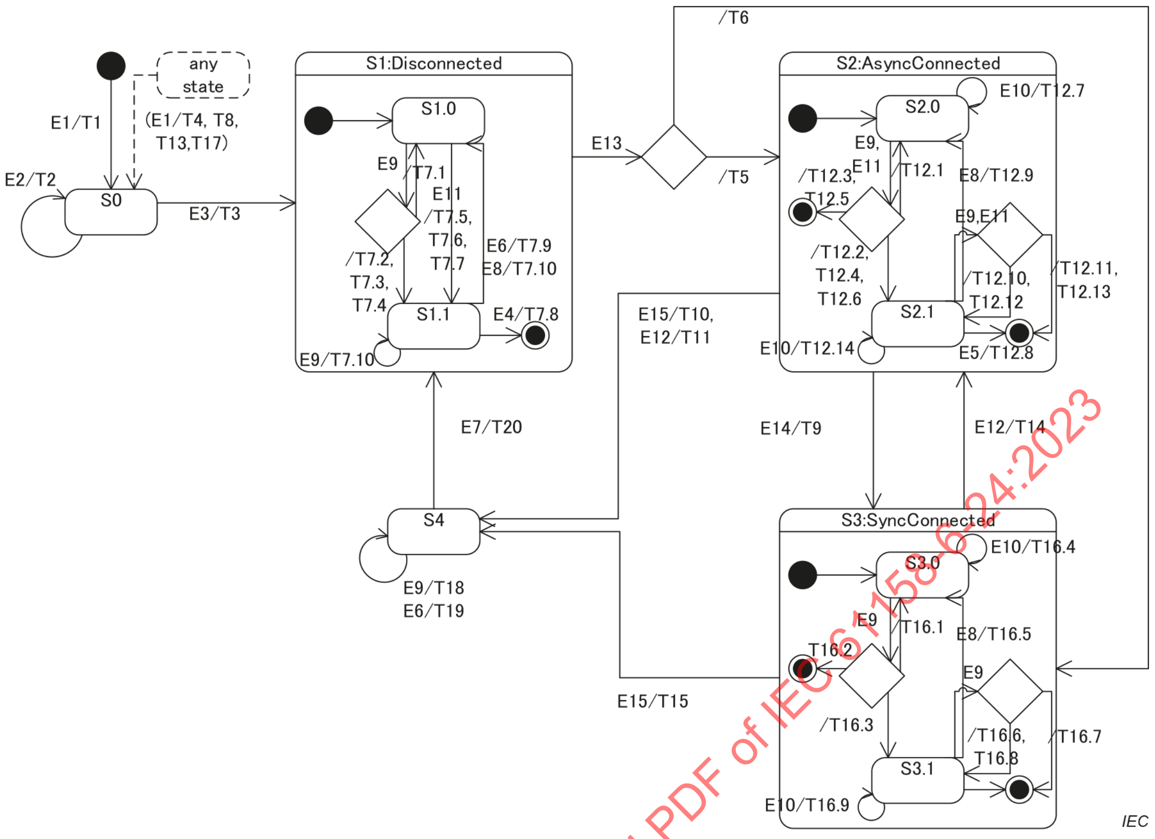


Figure 19 – Statechart diagram of FDCPM-S

Table 13 – State descriptions of FDCPM-S

S#	State	Substate	Description
S0	Disabled	-	State when the Slave Class of FDC ASE (FDC Slave) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from FSM ASE. Entry/ Initial values are set in Attributes of the Slave object.
S1	Disconnected	-	State when the connection is released and the PM is waiting Connect.request from the peer C1 master AP In the lower layer, a communication has been started. Only the connection control commands (CONNECT, DISCONNECT) and NOP command is allowed to be transferred in this state. Even if the DL layer is running in the cyclic communication mode, the communication cycle has not been notified in this state because no connection is established.
S1.0		Idle	Substate when the PM is waiting to receive a CMD-PDU from a peer C1 master AP
S1.1		WaitRSPComplete	Substate when the PM is waiting for the corresponding response to transfer from a FAL user or the Slave AP.

S#	State	Substate	Description
S2	AsyncConnected		A normal operation state to transfer the controlling command from the C1 Master AP to the Slave AP and execute a function of a field device with the asynchronous type command A connection with the FDC Master is established and a communication cycle event is notified by AR ASE. Any asynchronous type command can be transferred in this state. No synchronous type command is allowed to be transferred.
S2.0		Idle	Substate when the PM is waiting to receive a CMD-PDU from a peer C1 master AP
S2.1		WaitRSPComplete	Substate when the PM is waiting for the corresponding response to transfer from a FAL user or the Slave AP.
S3	SyncConnected	-	The most time-critical operation state to transfer the controlling command from the C1 Master AP to the Slave AP and execute a function of a field device with both synchronous and asynchronous commands A connection with the FDC Master has been established and a communication cycle event is notified by AR ASE. That event is transferred as FDC-ComCycle.ind to the Slave AP. The WDT counter on every CMD-PDUs shall function to watch over the status of the synchronization activity of the C1 master AP. And so as to the RWDT counter on RSP-PDUs for the Slave AP.
S3.0		Idle	Substate when the PM is waiting to receive a CMD-PDU from a peer C1 master AP
S3.1		WaitRSPComplete	Substate when the PM is waiting for the corresponding response to transfer from a FAL user or the Slave AP.
S4	Disconnecting	-	State when the connection changes to be released because communication errors continuously occur or the FAL user demands it. The PM waits in order to prevent the system malfunctioning as a kind of interlocking mechanism, and then recovers to S1 state when receiving a resume request from the user (ResumeCycle.req).

8.2.5.2 Triggering events

Table 14 lists each trigger event of the FDCPM-S.

Table 14 – Trigger event descriptions of FDCPM-S

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FDC-Reset.req	FSM ASE		
E2	FDC-Open.req	FSM ASE	AREPID	
E3	FDC-Enable.req	FSM ASE	TransmissionMode	
E4	FDC-Connect.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	
E5	FDC-SyncSet.rsp	FAL user (Salve AP)	ServiceStatus, ProgressStatus, RSP-SDU	
E6	FDC-Disconnect.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E7	FDC-ResumeCycle.req	FAL user (Slave AP)		
E8	FDC-Command.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	
E9	FDC-DataExchange.req	AR ASE (FDCS-AR)	CMD-SDU	
E10	AR-CycleEvent.ind	AR ASE (FDCS-AR)	NetworkClock	
E11	AR-SendCommand.ind	AR ASE (FDCS-AR)	CMD-SDU	
E12	Error detected	This object		See 8.2.7. WDT failures are detected twice serially under SyncConnect state. Alternatively, failures are detected continuously under AsyncConnect state.
E13	EventConnectRSP	This object (submachine)	status Unsigned16 rsdu _CONNECT-RSP-PDU	Notification from submachine
E14	EventSyncSetRSP	This object (submachine)	status Unsigned16 rsdu _SYNC_SET-RSP-PDU	Notification from submachine
E15	EventDisconnectCMD	This object (submachine)	csdu _DISCONNECT-CMD-PDU	Notification from submachine

8.2.5.3 Action descriptions at state transitions

Table 15 describes state transitions of the main SM of FDCPM-S. Moreover,

Table 16 describes state transitions of the submachine of FDCPM-S.

Table 15 – Transitions of main SM of FDCPM-S

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC slave --*/;	S0:Disabled
T2	S0:Disabled	E2:FDC-Open.req / /*-- Initializing the FDC slave --*/; [a]	S0:Disabled
T3	S0:Disabled	E3:FDC-Enable.req (communicationMode) / TransMode = communicationMode;	S1:Disconnected
T4	S1:Disconnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Disabled

T#	Source State	Event (arguments) [conditions] / action	Target State

T5	S1:Disconnected	E13:EventConnectRSP (status, rsdu) [rsdu.rspBody.syncmode != 1]
		/ AR-StartComCycle.req;
		ProtocolVersion = rsdu.rspBody.ver.;
		SyncMode = 0;
		DTMode = rsdu.rspBody.dtmode;
		SubCMDMode = rsdu.rspBody.subcmd;
		ComTime = rsdu.rspBody.com_time;
		DevProfileType = rsdu.rspBody.profile_type;
			S2:AsyncConnected

T6	S1:Disconnected	E13:EventConnectRSP (status, rsdu) [rsdu.rspBody.syncmode == 1]
		/ AR-StartComCycle.req;
		ProtocolVersion = rsdu.rspBody.ver;
		SyncMode = 1;
		DTMode = rsdu.rspBody.dtmode;
		SubCMDMode = rsdu.rspBody.subcmd;
		ComTime = rsdu.rspBody.com_time;
		DevProfileType = rsdu.rspBody.profile_type;
			S3:SyncConnected

T7	S1:Disconnected	The other events / /*-- state-transition in the submachine --*/;	S1:Disconnected
T8	S2:AsyncConnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC slave --*/;	S0:Disabled

T9	S2:AsyncConnected	E14:EventSyncSetRSP (status, rsdu)
		/
		WDT.LastMN = 0;
		WDT.LastSN = 0;
		RWDT.LastRMN = 0;
		RWDT.LastRSN = 0;
			S3:SyncConnected

T10	S2:AsyncConnected	E15:EventDisconnectCMD (csdu) / /*-- no-operation --*/;	S4:Disconnecting
T11	S2:AsyncConnected	E12: Errors occur continuously [b, c] / /*-- no-operation --*/;	S4:Disconnecting
T12	S2:AsyncConnected	The other events / /*-- state-transition in the submachine --*/;	S2:AsyncConnected
T13	S3:SyncConnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC master --*/;	S1:Disabled
T14	S3:SyncConnected	E12: Errors occur serially twice [b] / /*-- no-operation --*/;	S2:AsyncConnected
T15	S3:SyncConnected	E15:EventDisconnectCMD (csdu) / /*-- no-operation --*/;	S4:Disconnecting

T#	Source State	Event (arguments) [conditions] / action	Target State
T16	S3:SyncConnected	The other events / /*-- state-transition in the submachine --*/;	S3:SyncConnected
T17	S4:Disconnecting	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC slave --*/;	S1:Disabled

T18	S4:Disconnecting	E9:FDC-DataExchange.req (csdu)
		/ rsdu.cmd = disconnect;
		FDC-DataExchange.cnf (rsdu);
			S4:Disconnecting

T19	S4:Disconnecting	E6:FDC-Disconnect.rsp (rsdu)
		/ LastRSP-SDU = rsdu;
		If (TransMode == EventDriven)
		AR-SendCommand.rsp (rsdu);
		else /*-- no-operation --*/;
			S4:Disconnecting

T20	S4:Disconnecting	E7:FDC-ResumeCycle.req / /*-- Clearing internal information related to the communication cycle and alarm information --*/;	S2:Disconnected

[a] The detailed process depends on the system implementation.

[b] Implementers need some mechanisms to count or manage errors.

[c] The threshold of error duration count is an implementation matter.

Table 16 – Transitions of submachine of FDCPM-S

T#	Source State	Event (arguments) [conditions] / action	Target state

T7.1	S1.0:Idle	E9:FDC-DataExchnage.req (csdu) [csdu == LastCMD-SDU]
		/ rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
			S1.0: Idle

T7.2	S1.0:Idle	E9:FDC-DataExchnage.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == connect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Connect.ind (csdu);
			S1.1: WaitRSPComplet e

T7.3	S1.0:Idle	E9:FDC-DataExchnage.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Disconnect.ind (csdu);
			S1.1: WaitRSPComplet e

T#	Source State	Event (arguments) [conditions] / action	Target state

T7.4	S1.0:Idle	E9:FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == nop)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Command.ind (csdu);
			S1.1: WaitRSPComplet e

T7.5	S1.0:Idle	E11:AR-SendCommand.ind (status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == connect)]
		/ LastCMD-SDU = csdu;
		FDC-Connect.ind (csdu);
			S1.1: WaitRSPComplet e

T7.6	S1.0:Idle	E11:AR-SendCommand.ind (status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		FDC-Disconnect.ind (csdu);
			S1.1: WaitRSPComplet e

T7.7	S1.0:Idle	E11:AR-SendCommand.ind (status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == nop)]
		/ LastCMD-SDU = csdu;
		FDC-command.ind (csdu);
			S1.1: WaitRSPComplet e

T7.8	S1.1: WaitRSPComplete	E4: FDC-Connect.rsp (rsdu) / LastRSP-SDU = rsdu; If (TransMode == EventDriven) {AR-SendCommand.rsp (rsdu);/*-- signal E13:EventConnectRSP to the main machine --*/;} else /*-- no-operation --*/;	exit from the submachine

T7.9	S1.1:
	WaitRSPComplete
		E6:FDC-Disconnect.rsp (rsdu)
		/ LastRSP-SDU = rsdu;
		If (TransMode == EventDriven)
		{AR-SendCommand.rsp (rsdu);}
		else /*-- no-operation --*/;
			S1.0: Idle

T7.10	S1.1:
	WaitRSPComplete
		E8:FDC-Command.rsp (rsdu)
		/ LastRSP-SDU = rsdu;
		If (TransMode == EventDriven)
		{AR-SendCommand.rsp (rsdu);}
		else /*-- no-operation --*/;
			S1.0: Idle

T7.11	S1.1:
	WaitRSPComplete
		E9: FDC-DataExchnage.req (csdu)
		/ rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
			S1.1: WaitRSPComplet e

T12.1	S2.0: Idle	E9: FDC-DataExchange.req (csdu) [csdu == LastCMD-SDU]
		/ rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
			S2.0: Idle

T12.2	S2.0:Idle	E9:FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == sync_set)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-SyncSet.ind (csdu);
			S2.1: WaitRSPComplet e

T#	Source State	Event (arguments) [conditions] / action	Target state

T12.3	S2.0:Idle	E9:FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Disconnect.ind (csdu);
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			exit from the submachine

T12.4	S2.0: Idle	E9: FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && ((csdu.cmd != sync_set) || (csdu.cmd != disconnect))]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Command.ind (csdu);
			S2.1: WaitRSPComplete

T12.5	S2.0:Idle	E11:AR-SendCommand.ind (status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		FDC-Disonnect.ind (csdu);
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			exit from the submachine

T12.6	S2.0:Idle	E11:AR-SendCommand.ind (status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)]
		/ LastCMD-SDU = csdu;
		FDC-Command.ind (csdu);
			S2.1: WaitRSPComplete

T12.7	S2.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind;	S2.0:Idle
T12.8	S2.1: WaitRspComplete	E5: FDC-SyncSet.rsp (rsdu) / LastRSP-SDU = rsdu; /*-- signal E14:EventSyncSetRSP to the main machine --*/;	exit from the submachine

T12.9	S2.1: WaitRspComplete	E8: FDC-Command.rsp (rsdu)
		/ LastRSP-SDU = rsdu;
		If (TransMode == EventDriven)
		{AR-SendCommand.rsp (rsdu);}
		else /*-- no-operation --*/;
			S2.0: Idle

T12.10	S2.1: WaitRSPComplete	E9: FDC-DataExchange.req (csdu) [csdu == LastCMD-SDU]
		/ rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
			S2.1: WaitRSPComplete

T12.11	S2.1: WaitRSPComplete	E9:FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Disconnect.ind (csdu);
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			exit from the submachine

T#	Source State	Event (arguments) [conditions] / action	Target state

T12.12	S2.1: WaitRSPComplete	E9: FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Command.ind (csdu);
			S2.1: WaitRSPComplet e

T12.13	S2.1: WaitRSPComplete	E11:AR-SendCommand.ind (status, csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		FDC-Disonnect.ind (csdu);
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			exit from the submachine

T12.14	S2.1: WaitRSPComplete	E10: AR-CycleEvent.ind / FDC-ComCycle.ind;	S2.1: WaitRSPComplet e

T16.1	S3.0: Idle	E9: FDC-DataExchange.req (csdu) [csdu == LastCMD-SDU]
		/ LastCMD-SDU = csdu;
		if (csdu.wdt.mn != LastMN)
		{/*-- signal E12 watchdog counter error to the main machine --*/;}
		else {
		LastMN = csdu.wdt.mn;
		rsdu = LastRSP-SDU;LastRMN = LastMN;
		rsdu.rwdt.rmn = LastRMN;
		rsdu.rwdt.rsn = LastRSN;
		FDC-DataExchange.cnf (rsdu);}
			S3.0: Idle
			or
			Exit to the main machine, when E12 has been issued.

T16.2	S3.0: Idle	E9:FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Disconnect.ind (csdu);
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			xit from the submachine

T16.3	S3.0: Idle	E9: FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)]
		/ LastCMD-SDU = csdu;
		if (csdu.wdt.mn != LastMN)
		{/*-- signal E12 watchdog counter error to the main machine --*/;}
		else {
		LastMN = csdu.wdt.mn;
		rsdu = LastRSP-SDU;
		LastRMN = LastMN;
		rsdu.rwdt.rmn = LastRMN;
		rsdu.rwdt.rsn = LastRSN;
		FDC-DataExchange.cnf (rsdu);
		FDC-Command.ind (csdu);}
			S3.1: WaitRSPComplet e
			or
			Exit to the main machine, when E12 has been issued.

T#	Source State	Event (arguments) [conditions] / action	Target state
T16.4	S3.0:Idle	E10: AR-CycleEvent.ind / FDC-ComCycle.ind;	S3.0:Idle
T16.5	S3.1: WaitRspComplete	E8: FDC-Command.rsp (rsdu) / LastRSP-SDU = rsdu;	S3.0: Idle

T16.6	S3.1: WaitRSPComplete	E9: FDC-DataExchange.req (csdu) [csdu == LastCMD-SDU]
		/
		if (csdu.wdt.mn != LastMN)
		{/*-- signal E12 watchdog counter error to the main machine --*/;}
		else {
		LastMN = csdu.wdt.mn;
		rsdu = LastRSP-SDU;
		LastRMN = LastMN;
		rsdu.rwdt.rmn = LastRMN;
		rsdu.rwdt.rsn = LastRSN;
		FDC-DataExchange.cnf (rsdu);}
			S3.1: WaitRSPComplete
			or
			Exit to the main machine, when E12 has been issued.

T16.7	S3.1: WaitRSPComplete	E9:FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)]
		/ LastCMD-SDU = csdu;
		rsdu = LastRSP-SDU;
		FDC-DataExchange.cnf (rsdu);
		FDC-Disconnect.ind (csdu);
		/*-- signal E15:EventDisconnectCMD to the main machine --*/;
			exit from the submachine

T16.8	S3.1: WaitRSPComplete	E9: FDC-DataExchange.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd != disconnect)]
		/ LastCMD-SDU = csdu;
		if (csdu.wdt.mn != LastMN)
		{/*-- signal E12 watchdog counter error to the main machine --*/;}
		else {
		LastMN = csdu.wdt.mn;
		rsdu = LastRSP-SDU;
		LastRMN = LastMN;
		rsdu.rwdt.rmn = LastRMN;
		rsdu.rwdt.rsn = LastRSN;
		FDC-DataExchange.cnf (rsdu);
		FDC-Command.ind (csdu);}
			S3.1: WaitRSPComplete
			or
			Exit to the main machine, when E12 has been issued.

T16.9	S3.1: WaitRSPComplete	E10: AR-CycleEvent.ind
		/FDC-ComCycle.ind;
		/*-- counting up watchdog counter --*/;
		LastMN++;
		LastRSN++;
			S3.1: WaitRSPComplete

8.2.6 Monitor Protocol Machine (FDCPM-MN)

8.2.6.1 State descriptions

Figure 20 shows the FDCPM-MN statechart diagram, and Table 17 describes each state of the FDCPM-MN.

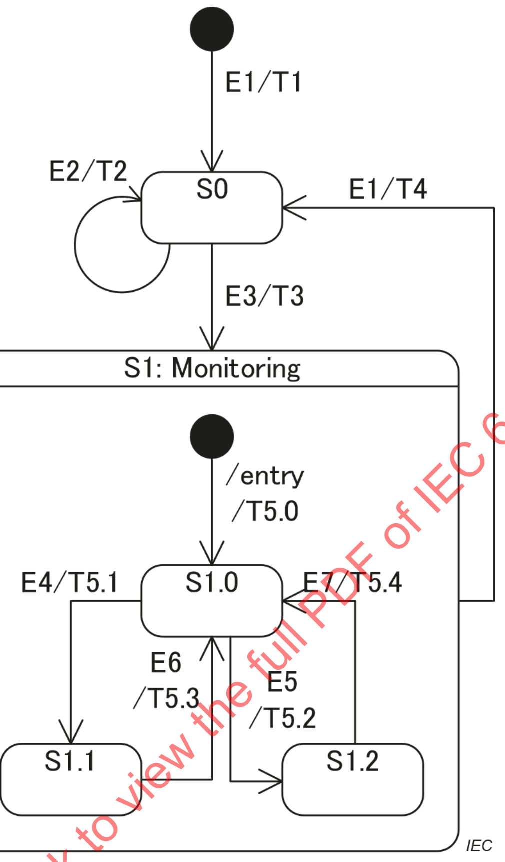


Figure 20 – Statechart diagram of FDCPM-MN

Table 17 – State descriptions of FDCPM-MN

S#	State	Substate	Description
S0	Disabled		State when the Monitor Class of FDC ASE (FDC Monitor) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. Entry/ Initial values are set in Attributes of the Monitor object.
S1	Monitoring		A normal operation state to provide services to monitor a CMD-PDU or a RSP-PDU for the FAL user or the Monitor AP.
S1.0		Idle	Substate when the PM is waiting the read request for a CMD-PDU or a RSP-PDU from a FAL user.
S1.1		WaitCMDData	Substate when the PM is waiting to get a CMD-PDU from the corresponding AR ASE
S1.2		WaitRSPData	Substate when the PM is waiting to get a RSP-PDU from the corresponding AR ASE

8.2.6.2 Triggering events

Table 18 lists each trigger event of the FDCPM-MN.

Table 18 – Trigger event descriptions of FDCPM-MN

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	FDC-Reset.req	FSM ASE		
E2	FDC-Open.req	FSM ASE	AREPIDList	
E3	FDC-Enable.req	FSM ASE	TransmissionMode	
E4	FDC-GetCMD.req	FAL user (Monitor AP)	APID	
E5	FDC-GetRSP.req	FAL user (Monitor AP)	APID	
E6	AR_GetCMD.cnf	AR ASE (FDCMN-AR)	ServiceStatus, CMD-SDU	
E7	AR-GetRSP.cnf	AR ASE (FDCMN-AR)	ServiceStatus, RSP-SDU	

8.2.6.3 Action descriptions at state transitions

Table 19 describes state transitions of the main SM of FDCPM-MN. Moreover, Table 20 describes state transitions of the submachine of FDCPM-MN.

Table 19 – Transitions of main SM of FDCPM-MN

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(Initial state)	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC monitor --*/;	S0:Disabled
T2	S0:Disabled	E2:FDC-Open.req / /*-- Initializing the FDC monitor --*/; [a]	S0:Disabled
T3	S0:Disabled	E3:FDC-Enable.req (communicationMode)/ TransMode = communicationMode; /*-- Stores the communication mode into the attribute.--*/	S1:Monitoring
T4	S1:Monitoring	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC monitor --*/;	S0:Disabled
[a] The detailed process depends on the system implementation.			

Table 20 – Transitions of submachine of FDCPM-MN

T#	Source State	Event (arguments) [conditions] / action	Target state
T5.0 (T3)	S0	/entry / /*-- no-operation --*/;	S1.0:Idle
T5.1	S1.0:Idle	E4:FDC-GetCMD.req (NodeID) / /*-- Generates DL-NodeID from the NodeID --*/ AR-GetCMD.req (DL-NodeID)	S1.1:WaitCMDData
T5.2	S1.0:Idle	E5:FDC-GetRSP.req (NodeID) / /*-- Generates DL-NodeID from the NodeID --*/. AR-GetRSP.req (DL-NodeID)	S1.2:WaitRSPData
T5.3	S1.1:WaitCMDDat a	E6:AR-GetCMD.cnf (+) (CMD_ASDU) / FDC-GetCMD.cnf (+) (CMD_ASDU)	S1.0:Idle
T5.4	S1.2:WaitRSPDat a	E7:AR-GetRSP.cnf (+) (RSP_ASDU) / FDC-GetCMD.cnf (+) (RSP_ASDU)	S1.0:Idle

8.2.7 Error procedure summary

8.2.7.1 Error detecting overview

The following shows the errors detected in the FDC service. When an error is detected, if any command is executed, both of the master and slave shall stop it and transition into a state where it can accept a new command to execute the resume process performed by the AP.

Only the definition of the errors is described in this document. The primary error detection processing or mechanism depends on the implementation, and so do the alarm notifying method and the resume processing from the error. They are out of the scope of this document.

The following errors are defined:

- communication error: receipt status such as no DL-PDU receipt and FCS error detected in the lower layer;
- watchdog counter error: abnormal value of the watchdog counter of the partner in synchronous communication state, which is detected in the mutual monitoring by the master and the slave;
- illegal cycle timing: abnormal value (beyond the jitter allowance) in the transmission cycle detected by the slave;
- response timeout: response time-out detected by the master;
- illegal sync command: synchronization command reception in the state other than the synchronous communication, which is detected by the slave user.

8.2.7.2 Communication error

FDC master and slave can receive a notification in FDC-DataExchange request from AR ASE when FCS error or no-reception error for the reception DL-PDU is detected by the lower layer.

If the PM is under the synchronous communication state when the error is detected, it shall make a state transition to the asynchronous communication state.

In case of detection by an FDC master PM, if it has a pending command and waiting the response, it shall discard the command to abort the waiting state.

In case of detection by an FDC slave PM, if it has a processing command, it shall discard the command and the just sending response as well, if it has one. And the FDC slave PM shall edit the cmd_stat field or the status field in the RSP-PDU.

8.2.7.3 Watchdog counter error

The watchdog counters shall be a mechanism in the synchronous communication state, for one FDC ASE to detect the peer ASE's out of sync with the communication cycle. Once the ASE detects the illegal counter value in a received FDCService-PDU, it shall alert the error to the FDC user through the FSM ASE.

When either of an FDC master PM or an FDC slave PM detects the error serially twice, it shall make the state transition to an asynchronous communication state, not to accept synchronous commands after that.

In the case of detection by the FDC Master PM, if the command is pending and the FDC Master PM is waiting for a response, it shall discard the command to abort the waiting state.

In case of detection by an FDC slave PM, if it has a processing command, it shall discard the command and the just sending response as well, if it has one. And the FDC slave PM shall edit the cmd_stat field or the status field in the RSP-PDU.

8.2.7.4 Illegal cycle timing

The "illegal cycle timing" is a kind of error that shows that an FDC slave detects an abnormal cycle jitter by measuring the transmission cycle period. At every transmission cycle events, the FDC slave shall measure the elapsed time since the previous event. And if the measured value is different from the pre-defined parameter for the transmission cycle, the slave shall alert the error to the FDC user through the FSM ASE.

When the FDC slave PM detects an error in the synchronous state, it makes a state transition to the asynchronous communication state, and then it shall not accept synchronous commands. And the FDC slave PM shall edit the cmd_stat field or the status field in the RSP-PDU.

If the slave has a processing command, it shall discard the command and the just sending response as well, if it has one.

8.2.7.5 Response timeout

The "response timeout" is a kind of error that shows that an FDC master detects no response from a FDC slave. After sending a command, the FDC master measures the elapsed time since a falling edge of cmdRdy bit in receiving each RSP-PDUs until the bit change to on (‘1'B), which means completion of the command processing of the slave. And if the time exceeds the pre-defined time limit, the master shall alert the error to the FDC user through the FSM ASE.

When the FDC master PM detects the error, it shall discard the command and abort the waiting state. The recovery process for both master and slave are implementation dependent.

8.2.7.6 Illegal Sync command

The "illegal sync command" is a kind of error that shows that an FDC slave receives a synchronous command in the wrong state other than the synchronous communication state.

When the FDC slave PM detects the error, it shall discard the command and execute alternative command pre-defined in implementation specifications for the device. And the FDC slave PM shall edit the rcmd field, and the cmd_stat field or the status field in the RSP-PDU.

8.2.7.7 Alarm through RSP-PDU.CMD_STAT

The error described above means an abnormal condition or malfunction for communication activities, whereas another trouble on a remote device should be informed to the FAL user, too.

Those statuses can be transferred as an alarm or a warning through the status fields in the short PDU type responses or the cmd_stat field in the enhanced PDU type responses. The status changes have no effect on the PM directly.

The alarm means a status to tell that the field device has detected a fatal problem to be solved and cannot continue normal working. And the warning means a status to tell that the device has detected a slight or passing problem but is still working normally.

8.3 Message Protocol Machine (MSGPM)

8.3.1 Protocol overview

8.3.1.1 General

In the Message service, two types of protocols are defined in the Type 24 FAL. One is a user message service that transfers message data by using requester-responder relationship and another is a one-way message service that handles a message from the sender to the accepter. An FAL user can execute message communication with a peer FAL user by using both of the user message and the one-way message.

8.3.1.2 Requester-responder type message (user message)

The user message communication shall be processed according to the PDU flow diagram in Figure 21.

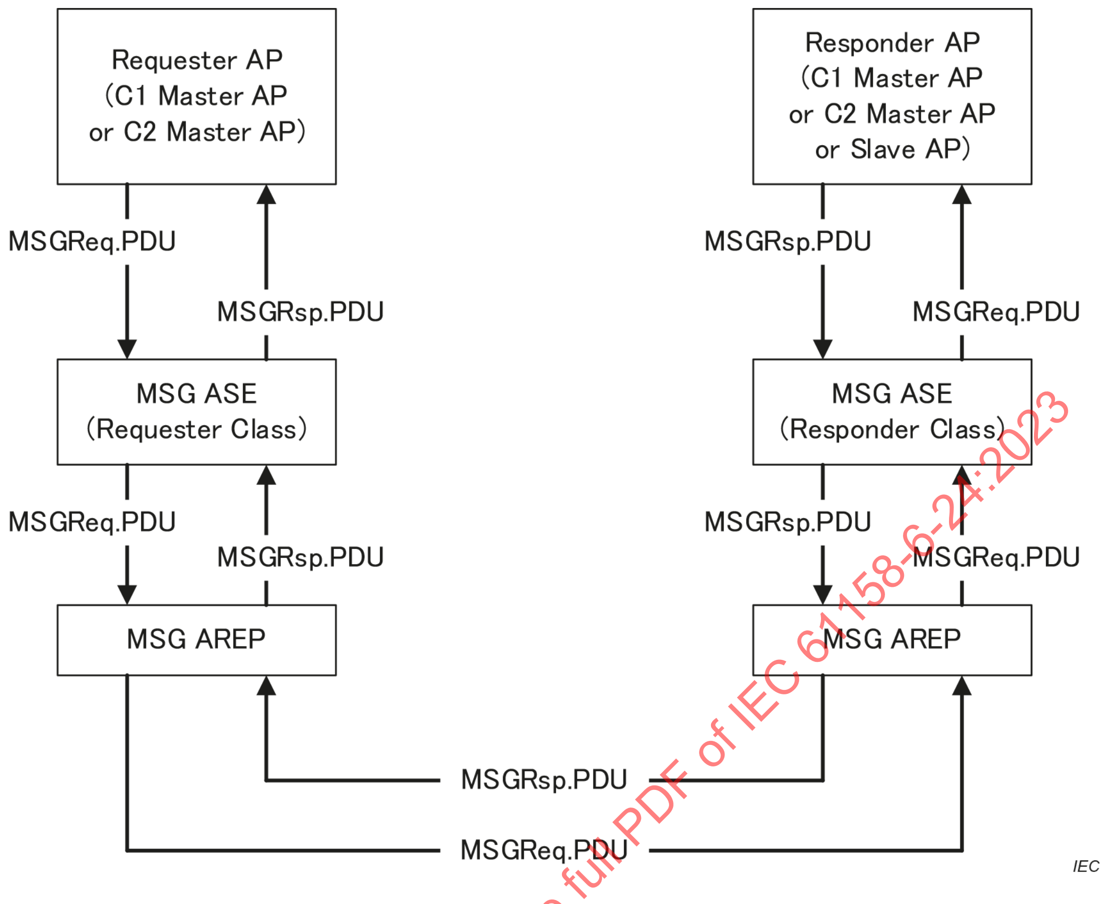

Requester AP (C1 Master AP or C2 Master AP)
Responder AP (C1 Master AP or C2 Master AP or Slave AP)
MSGReq.PDU
MSGRsp.PDU
MSG ASE (Requester Class)
MSG ASE (Responder Class)
MSG AREP
MSGRsp.PDU
MSGReq.PDU
IEC

Figure 21 – PDU transmission flow for user message

A message communication is started from a Requester AP.

The Requester AP sends an MSGReq-PDU to a Responder AP via an MSG ASE and an MSG AREP in the Requester side.

The Responder AP receives the MSGReq-PDU via an MSG AREP and an MSG ASE in the Responder side.

The Responder AP decodes the MSGReq-PDU, executes some activity according to the contents of the MSGReq-PDU, and generates the response data as an MSGRsp-PDU.

The MSGRsp-PDU is transferred to the Requester AP in the inverse process for the MSGReq-PDU.

The Requester AP decodes the MSGRsp-PDU and executes some activity according to the contents of it.

8.3.1.3 One-way message

A message communication that requires no response shall be processed according to the PDU flow diagram in Figure 22.

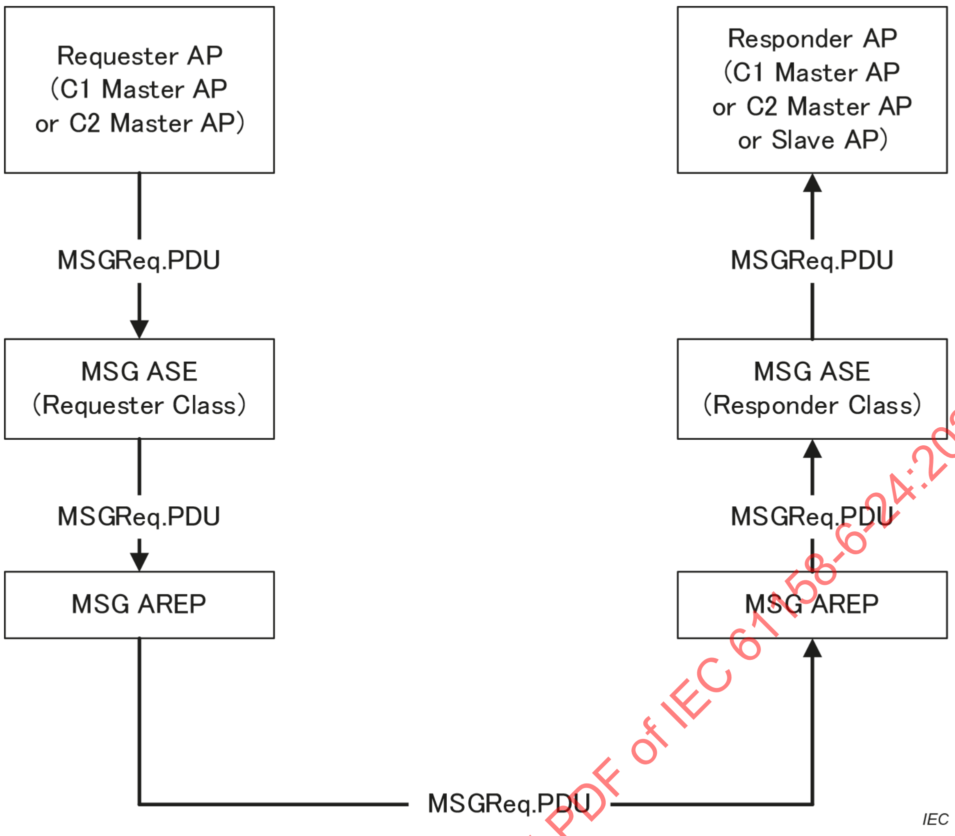


Figure 22 – PDU transmission flow for one-way message

A message communication is started from a Requester AP.

The Requester AP sends a MSGReq-PDU to a Responder AP via an MSG ASE and an MSG AREP in the Requester side.

The Responder AP receives the MSGReq-PDU via an MSG AREP and an MSG ASE in the Responder side.

The Responder-AP decodes the MSGReq-PDU and executes some activity according to the contents of it.

8.3.2 Requester Protocol Machine (MSGPM-RQ)

8.3.2.1 State descriptions

Figure 23 shows the MSGPM-RQ statechart diagram, and Table 21 describes each state of the MSGAR-RQ.

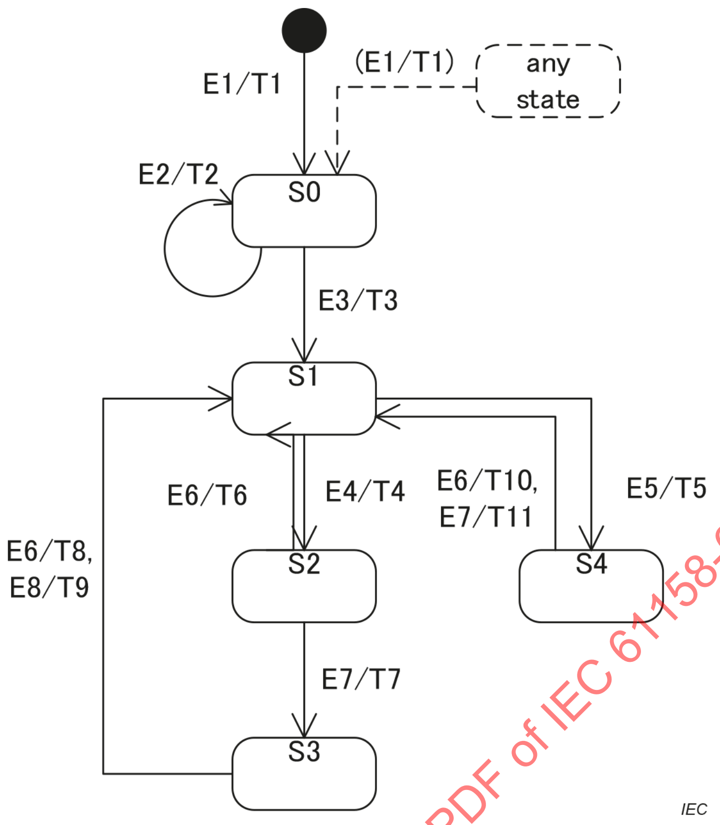


Figure 23 – Statechart diagram of MSGPM-RQ

Table 21 – State descriptions of MSGPM-RQ

S#	State	Substate	Description
S0	Disabled		State when the Requester Class of MSG ASE (Requester) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. --- Entry/ Initial values are set in Attributes of the MSG Requester object.
S1	WaitRequestMSG		State when the PM is waiting for a request from a FAL user to transfer a message
S2	SendingRequestMSG		State when the PM is transmitting the MSGREQ-PDU
S3	WaitComfirmMSG		State when the PM is waiting for the response from the peer Responder
S4	SendingOnewayMSG		State when the PM is transmitting the MSGREQ-PDU as a one-way message

8.3.2.2 Triggering events

Table 22 lists each trigger event of the MSGPM-RQ.

Table 22 – Trigger event descriptions of MSGPM-RQ

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	MSG-Reset.req	FSM ASE		
E2	MSG-Open.req	FSM ASE	AREPID	
E3	MSG-Enable.req	FSM ASE		
E4	MSG-UserMessage.req	FAL user (Requester AP)	TID, SPID, DPID, Length, MSGReq-SDU	
E5	MSG-OnewayMessage.req	FAL user (Requester AP)	TID, SPID, DPID, Length, MSGReq-SDU	
E6	MSG-AbortTransaction.req	FAL user (Requester AP)	TID	
E7	AR-SendMessage.cnf	AR ASE (MSG-AR)	ServiceStatus	
E8	AR-ReceiveMessage.cnf	AR ASE (MSG-AR)	ServiceStatus, SPID, Length, MSGService-SDU, Result	

8.3.2.3 Action descriptions at state transitions

Table 23 describes state transitions of the state machine of MSGPM-RQ.

Table 23 – Transitions of MSGPM-RQ

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1: MSG-Reset.req / /*-- Clearing all attributes of MSG requester --*/;	S0:Disabled
T2	S0:Disabled	E2: MSG-Open.req (AREPID) / /*-- Initializing MSG requester --*/; [a]	S0:Disabled
T3	S0:Disabled	E3: MSG-Enable.req / /*-- no-operation --*/;	S1:WaitRequestMSG
T4	S1:WaitRequestMSG	E4: MSG-UserMessage.req (SAPID, TID, Length, MSGReq-PDU) / AR-SendMessage.req (TID, Length, MSGReq-PDU);	S2:SendingRequestMSG
T5	S1:WaitRequestMSG	E5: MSG-OnewayMessage.req (SAPID, TID, Length, MSGReq-PDU) / AR-SendMessage.req (TID, Length, MSGReq-PDU);	S4:SendingOnewayMSG
T6	S2:SendingRequestMSG	E6: MSG-AbortTransaction.req (SAPID, TID) / AR-AbortMessage.req (SAPID, TID);	S1:WaitRequestMSG
T7	S2:SendingRequestMSG	E7: AR-SendMessage.cnf (+) / AR-ReceiveMessage.req;	S3:WaitComfirmMSG

T#	Source State	Event (arguments) [conditions] / action	Target State
T8	S3:WaitComfirmMSG	E6: MSG-AbortTransaction.req (SAPID, TID) / AR-AbortMessage.req (SAPID, TID);	S1:WaitRequestMSG
T9	S3:WaitComfirmMSG	E8: AR-ReceiveMessage.cnf (TID, Length, MSGRsp-PDU) / MSG-UserMessage.cnf (TID, Length, MSGRsp-PDU);	S1:WaitRequestMSG
T10	S4:SendingOnewayMSG	E6: MSG-AbortTransaction.req (SAPID, TID) /AR-AbortMessage.req;	S1:WaitRequestMSG
T11	S4:SendingOnewayMSG	E7: AR-SendMessage.cnf (Result) /MSG-OnewayMessage.cnf (TID);	S1:WaitRequestMSG
[a] The detailed process depends on the system implementation.			

8.3.3 Responder Protocol Machine (MSGPM-RS)

8.3.3.1 State descriptions

Figure 24 shows the MSGPM-RS statechart diagram, and Table 24 describes each state of the APCSM.

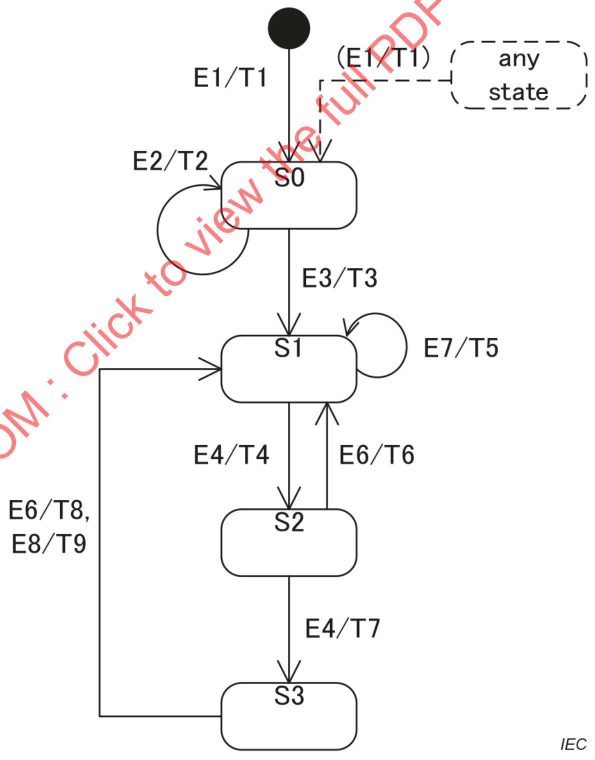


Figure 24 – Statechart diagram of MSGPM-RS

Table 24 – State descriptions of MSGPM-RS

S#	State	Substate	Description
S0	Disabled		State when the Responder Class of MSG ASE (Responder) has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. --- Entry/ Initial values are set in Attributes of the MSG Responder object.
S1	WaitIndicationMSG		State when the PM is waiting for a message from a Requester
S2	WaitResponseMSG		State when the PM is waiting for a response message from the FAL user (Responder AP)
S3	SendingResponseMSG		State when the PM is transmitting the response message to the Requester after receiving it from the Responder AP

8.3.3.2 Triggering events

Table 25 lists each trigger event of the MSGPM-RS.

Table 25 – Trigger event descriptions of MSGPM-RS

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	MSG-Reset.req	FSM ASE		
E2	MSG-Open.req	FSM ASE	AREPID	
E3	MSG-Enable.req	FSM ASE		
E4	MSG-UserMessage.rsp	FAL user (Responder AP)	ServiceStatus, TID, DPID, Length, MSGRsp-PDU	
E5	MSG-AbortTransaction.req	FAL user (Responder AP)	TID	
E6	AR-ReceiveMessage.cnf	AR ASE (MSG-AR)	ServiceStatus, SPID, Length, MSGService-PDU	
E7	AR-SendMessage.cnf	AR ASE (MSG-AR)	ServiceStatus	

8.3.3.3 Action descriptions at state transitions

Table 26 describes state transitions of the state machine of MSGPM-RS.

Table 26 – Transitions of MSGPM-RS

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1: MSG-Reset.req / /*-- Clearing all attributes of the MSG responder --*/;	S0:Disabled
T2	S0:Disabled	E2: MSG-Open.req / /*-- Initializing the MSG responder --*/;[a]	S0:Disabled
T3	S0:Disabled	E3: MSG-Enable.req / AR-ReceiveMessage.req;	S1:WaitIndicationMSG
T4	S1:WaitIndicationMSG	E7: AR-ReceiveMessage.cnf (TID, Length, MSGReq-PDU, Result) [Message that requires no response[b]] / MSG-OnewayMessage.ind (TID, Length, MSGReq-PDU);	S1:WaitIndicationMSG
T5	S1:WaitIndicationMSG	E7: AR-ReceiveMessage.cnf (TID, Length, MSGReq-PDU, Result) [Message that requires a response[b]] / MSG-UserMessage.ind (TID, Length, MSGReq-PDU);	S2:WaitResponseMSG

T6	S2:WaitResponseMSG	E6: MSG-AbortTransaction.req (SAPID, TID)
		/
		AR-AbortMessage.req;
		AR-ReceiveMessage.req;
			S1:WaitIndicationMSG

T7	S2:WaitResponseMSG	E4: MSG-UserMessage.rsp (TID, Length, MSGRsp-PDU) / AR-SendMessage.req (TID, Length, MSGRsp-PDU);	S3:SendingResponseMSG

T8	S3:SendingResponseMSG	E6: MSG-AbortTransaction.req (SAPID, TID)
		/
		AR-AbortMessage.req;
		AR-ReceiveMessage.req;
			S1:WaitIndicationMSG

T9	S3:SendingResponseMSG	E8: AR-SendMessage.cnf / AR-ReceiveMessage.req;	S1:WaitIndicationMSG

[a] The detailed process depends on the system implementation.

[b] Whether a response is required or not is defined according to the contents of the MSGReq-PDU. However, the details depend on the implemented user message specifications above the FAL.

9 Application relationship protocol machine (ARPM)

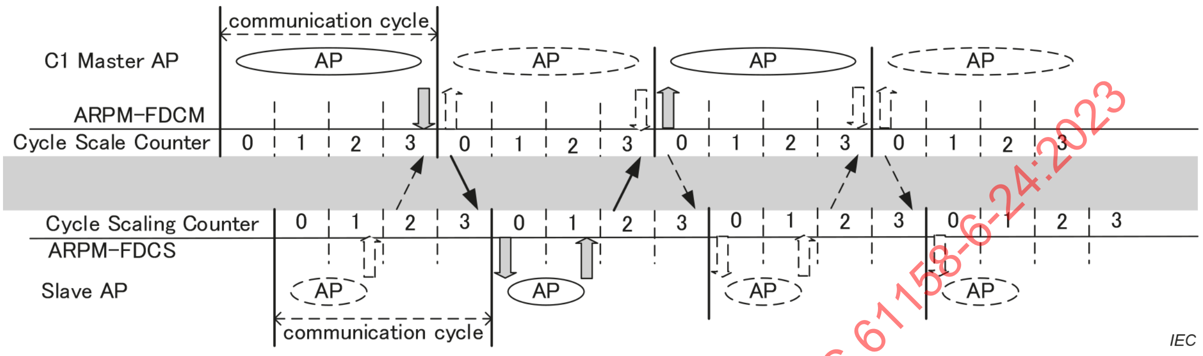
9.1 General

ARPM is the state machine for the AR ASE. It shall handle the lower layer to transmit and receive APDUs to and from the peer ASE. And it shall exchange those APDUs with other ASEs through providing the AR services in the local FAL.

9.2 ARPM for FDC ASE

9.2.1 Overview

This type of ARPM shall handle state-transition to exchange of a CMD-PDU and a RSP-PDU with the peer ARPM, shall manage a communication cycle and shall control the transfer mode (single transfer or dual transfer), as shown in Figure 25 and Figure 26. But it can neither manage the connection, the command transaction and the WDT values nor check the validity of contents of a CMD-PDU and a RSP-PDU.



NOTE The figure shows the case of $T_{\mathrm{COMCYC}}=T_{\mathrm{CYC}}\times4$,

where

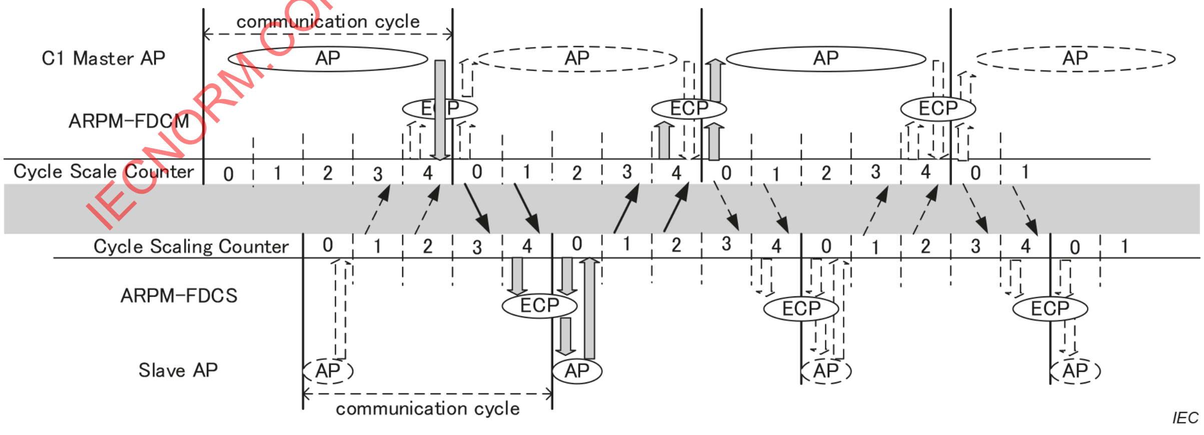
T_{COMCYC} is the communication cycle;

T_{CYC} is the transmission cycle.

Figure 25 – Example of single transfer process

FDC ASE can optionally provide the function of the dual transfer mode to improve the reliability of the received APDU. This mode is a communication method in which an identical APDU is transferred twice within one communication cycle, and then they are checked and verified on the receiver side to detect an error or missing PDU, see Figure 26.

The dual transfer mode can be operated only under the synchronous communication state and $T_{\mathrm{COMCYC}}\geq T_{\mathrm{CYC}}\times2$, where T_{COMCYC} is a communication cycle and T_{CYC} is a transmission cycle.



NOTE The figure shows the case of $T_{\mathrm{COMCYC}}=T_{\mathrm{CYC}}\times5$,

where

T_{COMCYC} is the communication cycle;

T_{CYC} is the transmission cycle;

ECP is the Error-detection and PDU-comparison process.

Figure 26 – Example of dual transfer process

Figure 27 shows the example of the application where the slave AP sends the APDU according to AR-CyclicEvent.ind of the protocol machine FDCPM-S. Receiving the notification of AR-CyclicEvent.ind, the slave AP responds APDU to the C1 master AP in the Communication Cycle. The C1 master AP receives APUD sent from the slave AP in the next Communication Cycle. In this application, the C1 master AP can receive the response from the slave AP for each communication cycle.

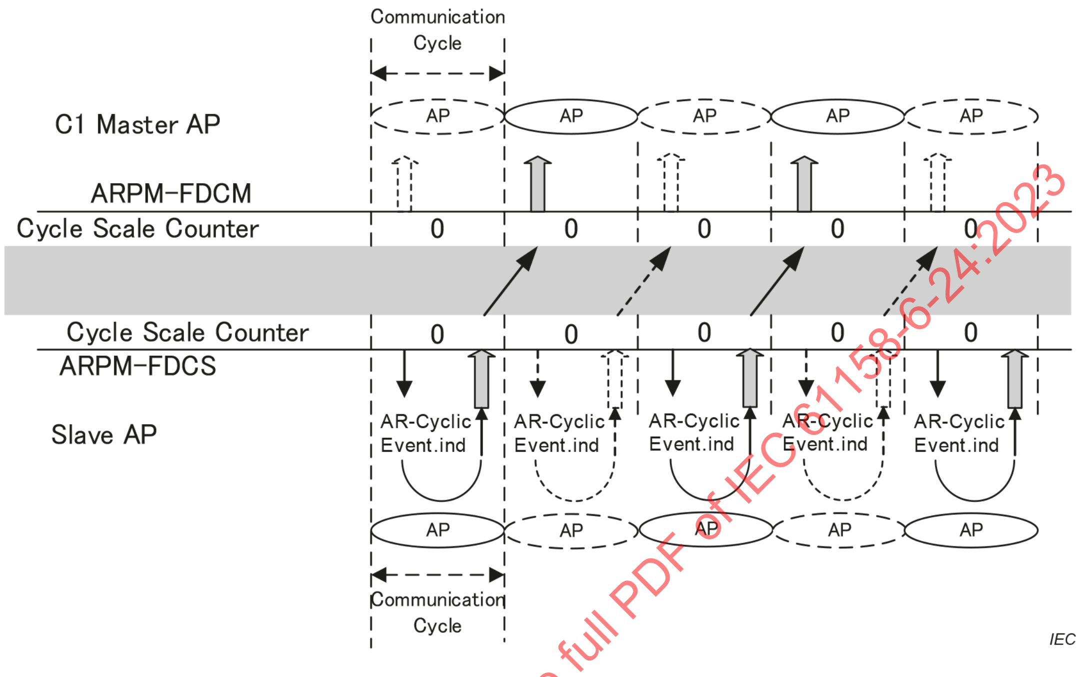


Figure 27 – Example of Synchronous command communication

Also, transmission cycles of integer multiple of the basic transmission cycle (Tcyc) can be set for each slave. This is the communication with multiple transmission cycles. For the communication with multiple transmission cycles, "individual transmission cycle multiple (tcc_mul)" and "individual transmission position number (tcc_pos)" shall be set for each slave.

Tcc_mul indicates the transmission cycle in which the slave returns a response. Tcc_pos indicates the transmission position number at which the slave returns a response. The C1 master shall send the command frame with the Cycle Counter setting, which is to be incremented for each transmission cycle. The slaves shall divide the Cycle Counter value with tcc_mul value set for the slaves and output division remainder. If the division remainder matches with tcc_pos, the slaves shall respond to the C1 master.

Figure 28 shows the timing chart when the communication with multiple transmission cycles is separately set for each slave. As Slave #1 has the setting of tcc_mul = 1 and tcc_pos = 1, it responds (R#1) for all transmission cycles. As Slave #2 has the setting of tcc_mul = 2 and tcc_pos = 2, it responds (R#2) once every two transmission cycles and at the second transmission cycle. As Slave #3 has the setting of tcc_mul = 4 and tcc_pos = 3, it responds (R#3) once every four transmission cycles and at the third transmission cycle.

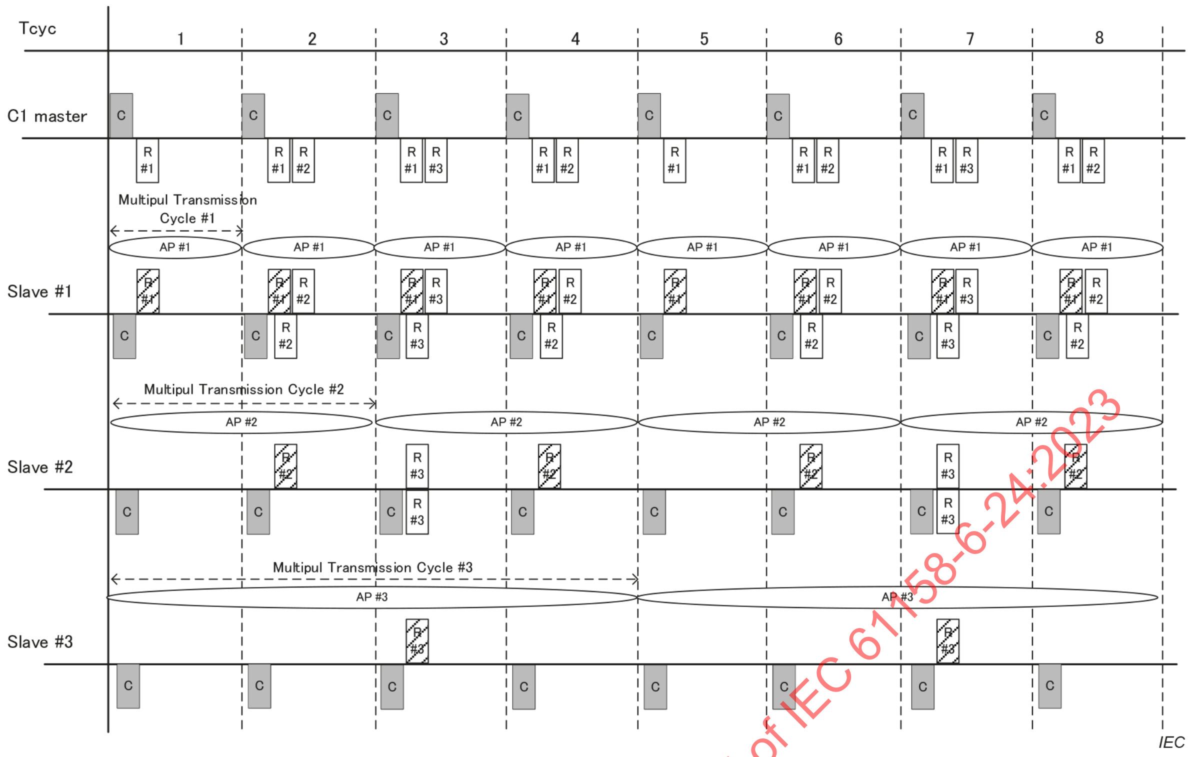

Tcyc
1
2
3
4
5
6
7
8
C1 master
C
R #1
R #2
R #3
Multipul Transmission Cycle #1
AP #1
Slave #1
Multipul Transmission Cycle #2
AP #2
Slave #2
Multipul Transmission Cycle #3
AP #3
Slave #3
IEC

Figure 28 – Timing chart for individual communication cycle setting

9.2.2 ARPM for FDC Master (ARPM-FDCM)

9.2.2.1 State descriptions

Figure 29 shows the ARPM-FDCM statechart diagram, and Table 27 describes each state of the ARPM-FDCM.

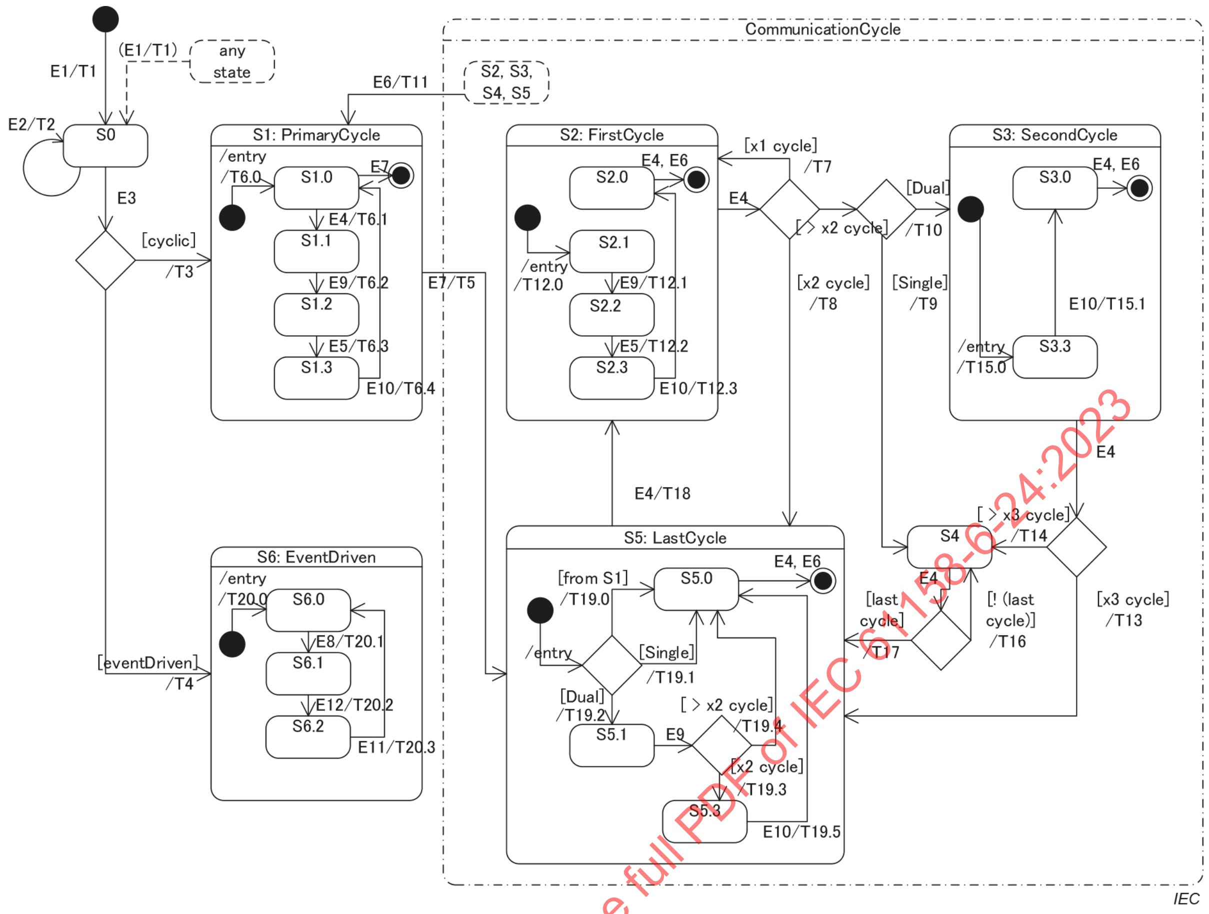


Figure 29 – Statechart diagram of ARPM-FDCM

Table 27 – State descriptions of ARPM-FDCM

S#	State	Substate	Description
S0	Disabled	-	State when the FDC Master AR Class has just been instantiated to an object as this PM The PM is waiting to finish creating an AP-context and to receive Enable.request from a FSM ASE. entry/ Initial values are set in Attributes of the FDCMaster AR object
S1	PrimaryCycle	-	State when a cyclic transmission mode in the lower layer is being executed in a specific transmission cycle and APDUs can be transmitted But the FDC PM is still in disconnected, and so no communication cycle has been generated yet but the just primary cycle has, which synchronize with the transmission cycle. entry/ Transition into the sub state: S1.0.
S1.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S1.1		WaitDL_GET_DATAcnf	Substate when the PM is waiting for the associated DL_GET_DATA.cnf after requesting the DL_GET_DATA.req to the related DL-SAPID
S1.2		WaitFDC-DataExchngecnf	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S1.3		WaitDL_SET_DATAcnf	Substate when the PM is waiting for the associated DL_SET_DATA.cnf after requesting the DL_SET_DATA.req to the related DL-SAPID

S#	State	Substate	Description
S2	FirstCycle	-	State when a connection in FDC ASE is established, the communication cycle is specified in the form of integer multiple of the primary cycle as ComTime, and in this first primary cycle the ARPM-FDCM exchanges PDUs between the FDC ASE and DLL If the communication cycle is the same as the primary cycle, the PM remains in this state while the FDC PM is in connected. entry/ Transition into the sub state: S2.1.
S2.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S2.1		WaitDL_GET_DATA.cnf	Substate when the PM is waiting for the associated DL_GET_DATA.cnf after requesting the DL_GET_DATA.req to the related DL-SAPID
S2.2		WaitFDC-DataExchnge.cnf	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S2.3		WaitDL_SET_DATA.cnf	Substate when the PM is waiting for the associated DL_SET_DATA.cnf after requesting the DL_SET_DATA.req to the related DL-SAPID
S3	SecondCycle	-	State when a connection in FDC ASE is established and the ARPM-FDCM is in the second primary cycle of the communication cycle If the communication cycle is twice primary cycle, the PM does not have a transition to this state but to LastCycle at the second primary cycle. In the dual transfer mode, the second CMD-PDU transmission is executed after the regular PDU exchange in the first cycle. entry/ Transition into the sub state: S3.3.
S3.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S3.3		WaitDL_SET_DATAcnf	Substate when the PM is waiting for the associated DL_SET_DATA.cnf after requesting the DL_SET_DATA.req to the related DL-SAPID
S4	MiddleCycle	-	State when a connection in FDC ASE is established and the ARPM-FDCM is in the middle primary cycle; not the 1st, the 2nd nor the last cycle If the communication cycle is treble primary cycle, the PM does not have a transition to this state but to LastCycle at the third primary cycle.
S5	LastCycle	-	State when a connection in FDC ASE is established and the ARPM-FDCM is in the last primary cycle of the communication cycle In the dual transfer mode, the PM receives a RSP-PDU from the slave and stores it into the internal buffer before the regular PDU exchange in the first cycle. entry [(the source state == S1) | (single transfer mode)] / Transition into the sub state: S5.0 entry [dual transfer mode] / Transition into the sub state: S5.1
S5.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S5.1		WaitDL_GET_DATAcnf	Substate when the PM is waiting for the associated DL_GET_DATA.cnf after requesting the DL_GET_DATA.req to the related DL-SAPID