IEC 61158-6-24

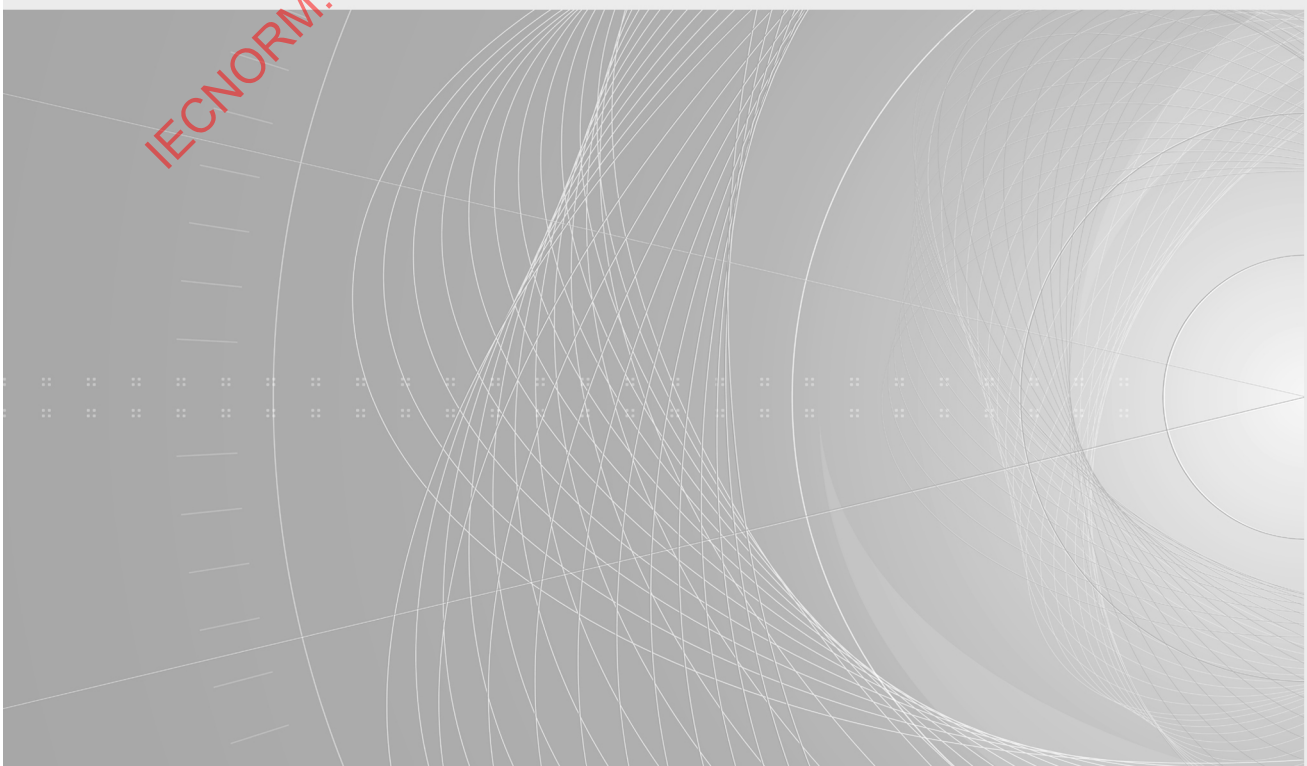
Edition 1.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-24: Spécification du protocole de la couche application – Éléments de type 24



IEC 61158-6-24:2014-08(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

IEC 61158-6-24

Edition 1.0 2014-08

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-24: Spécification du protocole de la couche application – Éléments de type 24

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XF

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-1769-6

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 IEC 61784-1 and IEC 61784-2.

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

The text of this standard is based on the following documents:

FDIS	Report on voting
65C/764/FDIS	65C/774/RVD

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

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

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 publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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- withdrawn,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

INTRODUCTION

This part of IEC 61158 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 standard 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 standard is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this standard together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems may 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 standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to 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 standard defines in an abstract way the externally visible behavior provided by the Type 24 fieldbus application layer in terms of

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

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

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

This standard 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 standard 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 IEC 61158-6.

1.3 Conformance

This standard 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, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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 IEC 61784-1 and IEC 61784-2 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 61158-5-24:2014, *Industrial communication networks – Fieldbus specifications – Part 5-24: Application layer service definition – Type 24 elements*

IEC 61158-6 (all parts), *Industrial communication networks – Fieldbus specifications – Part 6: Application layer protocol specification*

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 60559:2011, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

3 Terms, definitions, abbreviations, symbols and conventions

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

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 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 may be provided as a registered set of commands and responses or a 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 may 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
transfer
transmission
– communication: process to exchange information in a formal manner between two or more devices, users, APs or entities
– transfer: process to convey a PDU from a sender to a receiver
– transmission: process to send out and propagate electrical signals or encoded data

3.2.12
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 may synchronize with the transmission cycle multiplied by a specified scaling factor.

3.2.13
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

3.2.14
cyclic
repetitive in a regular manner

3.2.15
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.

3.2.16
cycle scale counter
counter to generate a communication cycle by means of scaling a primary cycle or a transmission cycle

3.2.17
device ID
part of “Device Information” to identify the device for a specific product type or model of the Type 24 fieldbus

3.2.18
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

3.2.19
device profile
collection of device-model-common information and functionality providing consistency between different device models among the same kind of devices

3.2.20
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

3.2.21
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.

3.2.22
error
abnormal condition or malfunction for communication or any other activities

3.2.23
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

3.2.24
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 f

3.2.25
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

3.2.26
message service
MSG service
communication service that handles the variable length data and not required a severe restriction on response time

3.2.27
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

3.2.28
monitor slave
variant of slave AP type who has both slave class and monitor class for FDC ASE of the Type 24 FAL

3.2.29
network clock
synchronized and periodically running counter that each nodes in a same network have, which becomes an oscillation source of the transmission cycle

3.2.30
primary cycle
period of repetitive activities synchronized with the transmission cycle before the connection establishing for the FDC protocol in Type 24 FAL

3.2.31
protocol machine
state machine that realizes a protocol as the main function of the entity in each layer

3.2.32
requester
class or its instance object of MSG ASE who plays a role of a command requester or sender for the Type 24 FAL

3.2.33
responder
class or its instance object of MSG ASE who plays a role of a command responder or receiver for the Type 24 FAL

3.2.34
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

3.2.35
service
operation or process that an object performs upon request from another object

3.2.36
single transfer
normal transfer mode for the FDC protocol of the Type 24 FAL in which a sender sends a PDU once a transaction

3.2.37
slave AP
AP type that has slave facilities for the FDC service of the Type 24 FAL, or the device implementing that AP type

3.2.38
slave
class or its instance object of FDC ASE who plays a role of a responder for the Type 24 FAL

3.2.39
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 may be provided as a registered set of commands and responses or a device profiles. See 4.4 and Annex A.

3.2.40
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.

3.2.41
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

3.2.42
transmission mode
state or a way of transmission for the lower layer protocol of the Type 24 fieldbus; cyclic mode, event driven mode

3.2.43
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.

3.2.44
warning
field device status to tell that the device has detected a slight or passing problem but 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 a implementation of each field devices.

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 standard.

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

This standard 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 the first character is "_" (low line) are used as data type symbols of FAL PDUs or their fields. While a same string as a PDU type but lacked the first "_" 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 may be used in the Type 24 FAL.

a) Basic types as simple types of ASN.1:

   i) INTEGER;

ii) REAL;

iii) BIT STRING;

iv) OCTET STRING;

v) NULL.

b) Specific basic types as subtypes from ASN.1:

vi) Integer8 ::= INTEGER (-128..127);

vii) Integer16 ::= INTEGER (-32 768 .. 32 767);

viii) Integer32 ::= INTEGER (-2 147 483 648 .. 2 147 483 647);

ix) Integer64 ::= INTEGER ((-1) × ‘8000 0000 0000 0000’H..‘7FFF FFFF FFFF FFFF’H);

x) Unsigned8 ::= INTEGER (0..’FF’H);

xi) Unsigned16 ::= INTEGER (0..’FFFF’H);

xii) Unsigend32 ::= INTEGER (0..’FFFF FFFF’H);

xiii) Unsigned64 ::= INTEGER (0..’FFFF FFFF FFFF FFFF’H);

xiv) Float32 ::= 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.

xv) Float64 ::= 0 | negative infinity | positive infinity

| REAL (WITH COMPONENTS { mantissa ($(-1)\times c_{64}$), 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.

xvi) IA5String

c) 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.

xvii) SEQUENCE

xviii) CHOICE

d) 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:

xix) SEQUENCE OF;

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

xxi) Array8 ::= SEQUENCE OF Integer8;

xxii) Array16 ::= SEQUENCE OF Integer16;

xxiii) Array32 ::= SEQUENCE OF Integer32;

xxiv) 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.

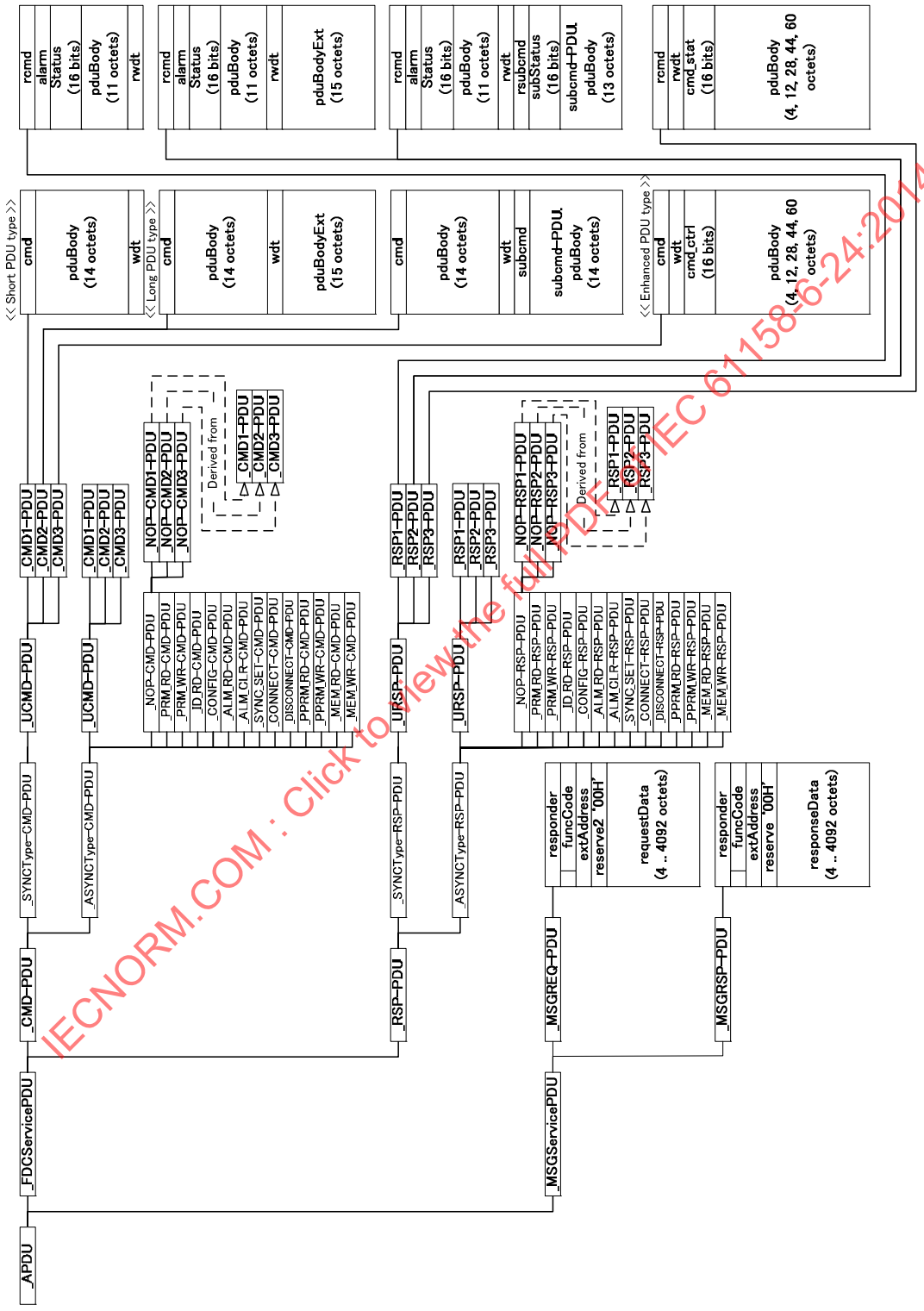


Figure 1 – Tree structure of APDU types

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 fourteen 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 may 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 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
_UCMD-PDU               ::= _CMD1-PDU | _CMD2-PDU | _CMD3-PDU
_NOP-CMD-PDU            ::= _NOP-CMD1-PDU | _NOP-CMD2-PDU | _NOP-CMD3-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
```

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 fourteen 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
_URSP-PDU           ::= _RSP1-PDU | _RSP2-PDU | _RSP3-PDU
_NOP-RSP-PDU        ::= _NOP-RSP1-PDU | _NOP-RSP2-PDU | _NOP-RSP3-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
```

4.2.2.2 Common component of PDU

4.2.2.2.1 FDC command PDU

_UCMD-PDU may have three types, short type: _CMD1-PDU, long type: _CMD2-PDU, and enhanced type: _CMD3-PDU. Both _CMD1-PDU and _CMD2-PDU shall have same header fields, but _CMD3-PDU shall have 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 able to be select from 4, 12, 28, 44, and 60 octets, but need not be switched on the fly.

```
_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) }
```

a) 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) }
```

b) 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 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 match 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.

c) 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 same as main cmd field in principle, but some codes may be 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) }
```

d) 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 may 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.

4.2.2.2.2 FDC response PDU

_URSP-PDU may have three types, short type: _RSP1-PDU, long type: _RSP2-PDU, and enhanced type: _RSP3-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 select from 4, 12, 28, 44, and 60 octets, but need not be switched on the fly.

```
_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
                                                                    PDU type
                                  rsubcmd   _SubCMD,
                                  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
                                                                    type
                       rcmd       _CMDCode,
                       rwdt       _RWDT,
                       cmd_stat   _CMD_STAT,
                       pduBody     OCTET STRING SIZE (4|12|28|44|60) }
```

a) 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.

b) 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
```

c) 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 fieldsto 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

d) 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 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 match the last rsn value plus one, it shall alert the watchdog counter error.

e) 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.

f) 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
```

g) 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 neither comm_alm nor cmd_alm.

Specification of device alarm status may depend 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 neither comm_alm nor cmd_alm.

Specification of device alarm status may depend 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 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 as follows.

'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 as follows.

'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.

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)}
```

a) 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.

b) 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.

c) 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.

d) 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.

e) requestData field and respondeData field

requestData and responseData
These fields shall contain the SDUs that may be 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) }
```

a) pNo field

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

The value range shall be from 0 to 65 535.

b) pSize field

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

The value range shall be from 0 to 255.

c) parameter field

parameter
This field shall contain the read data corresponding to pNo. The data size, data structure and its meaning may 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) }
```

a) pNo field

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

The value range shall be from 0 to 65 535.

b) pSize field

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

The value range shall be from 0 to 255.

c) parameter field

parameter
This field shall contain the data to write corresponding to pNo. The data size, data structure and its meaning may 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 may 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) }
```

a) 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.

b) 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.

c) idSize field

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

The value range shall be from 0 to 255.

d) idData field

idData
This field shall contain the read data corresponding to the idCode. The data size, data structure and its meaning may 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) } )
```

a) 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 may be 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) }
```

a) 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.

b) 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 solved the real causes or problems, to recover the device status to normal.

```
_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) }
```

a) 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 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

Exchange this 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 Unsigend8,
                                     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 Unsigend8,
                reserve    OCTET STRING SIZE (10)}
```

a) 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.

b) 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

Exchange this 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) }
```

a) pNo field

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

The value range shall be from 0 to 65 535.

b) pSize field

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

The value range shall be from 0 to 255.

c) parameter field

parameter
This field shall contain the data corresponding to pNo. The data size, data structure and its meaning may 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) }
```

a) pNo field

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

The value range shall be from 0 to 65 535.

b) pSize field

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

The value range shall be from 0 to 255.

c) parameter field

parameter
This field shall contain the data to write corresponding to pNo. The data size, data structure and its meaning may 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 Unsigend16,
                                pSize Unsigend8,
                                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 Unsigend16,
                                pSize Unsigend8,
                                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 may be 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) }
```

a) 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.

b) 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.

c) idSize field

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

The value range shall be from 0 to 255.

d) idData field

idData
This field shall contain the read data corresponding to the idCode. The data size, data structure and its meaning may 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) }
```

a) 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) }
```

a) 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.

b) 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.

c) 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)}
```

a) 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 Unsigend8
                                 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 Unsigend8
                                profile_type Unsigned8
                                reserve     OCTET STRING SIZE (0|8|24|40|56)}
```

a) 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 Unsigend16,
                                 pSize Unsigend8,
                                 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 Unsigend16,
                                 pSize Unsigend8,
                                 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) }
```

a) 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.

b) 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.

c) 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.

d) 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)  }
```

a) 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.

b) 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.

c) 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.

d) 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) } )
```

4.4 Device profile

A field device may 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.

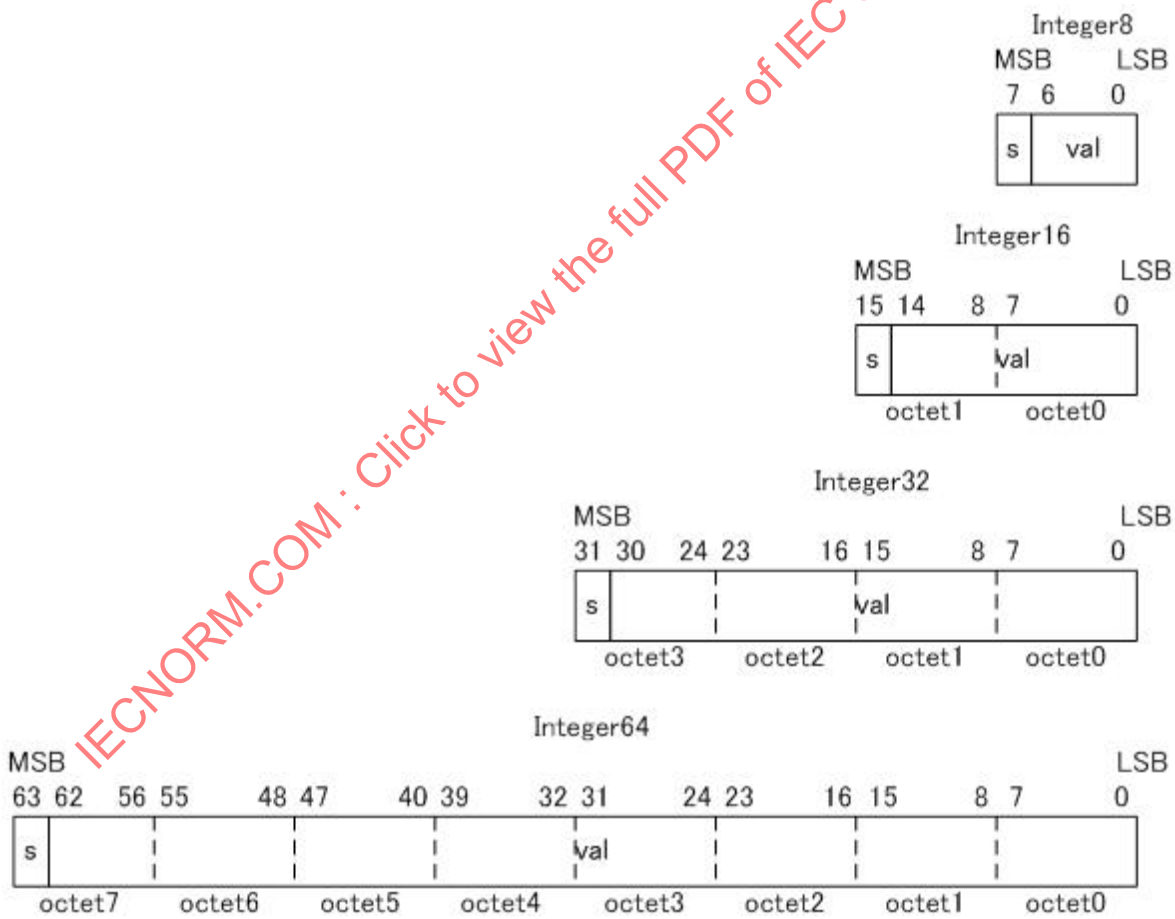
5.2 Encode rules

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 specifies 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).

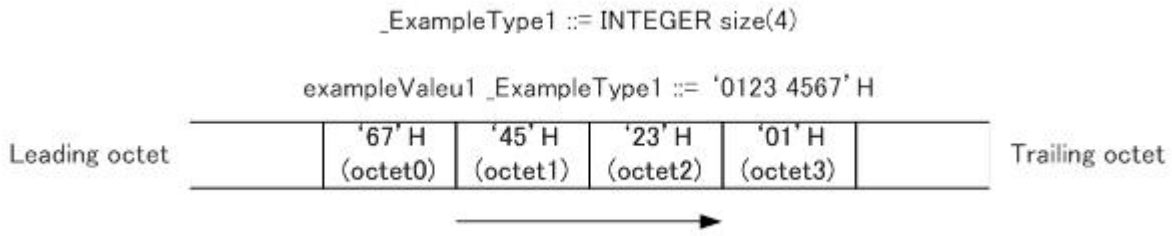


Figure 3 – Example of transfer of INTEGER value

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.

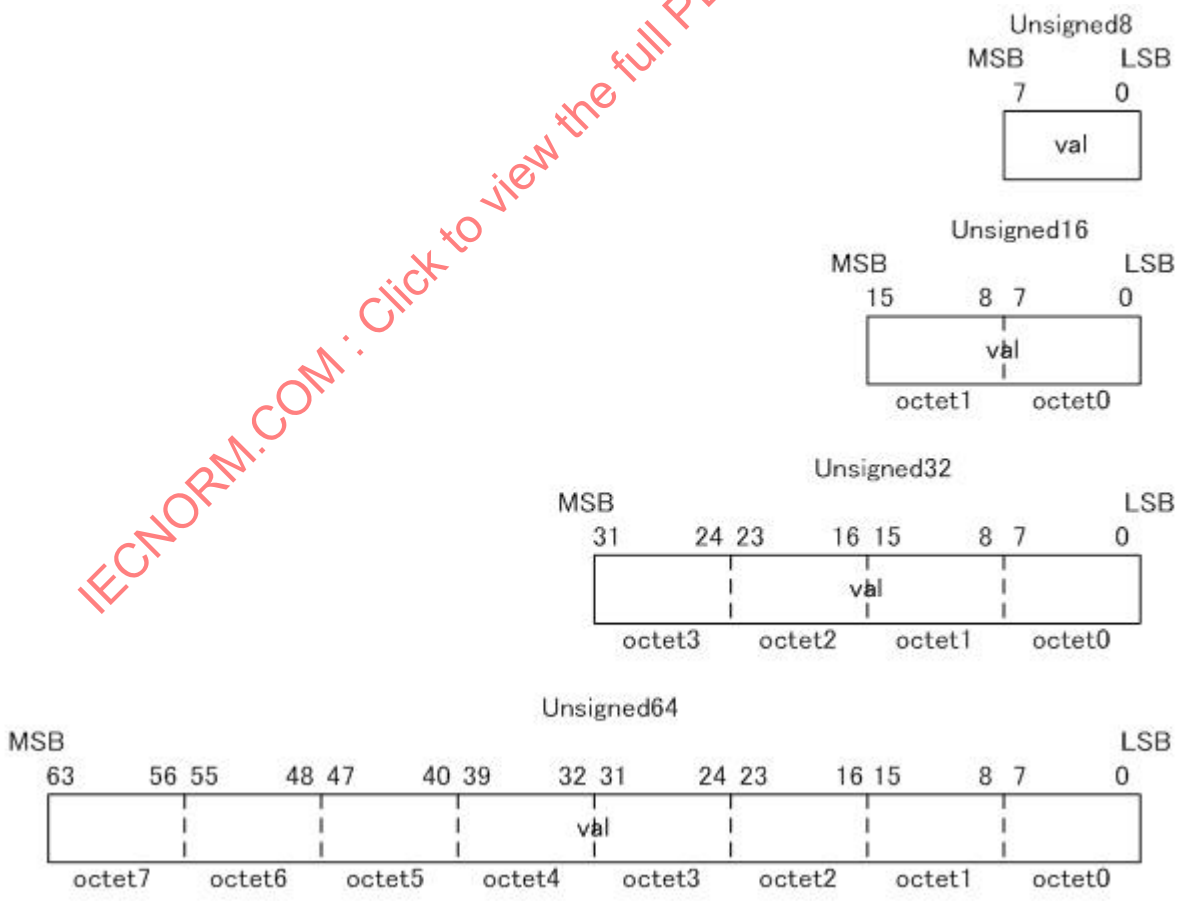


Figure 4 – Encode of Unsigned subtypes

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 Float32 and Float64 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.

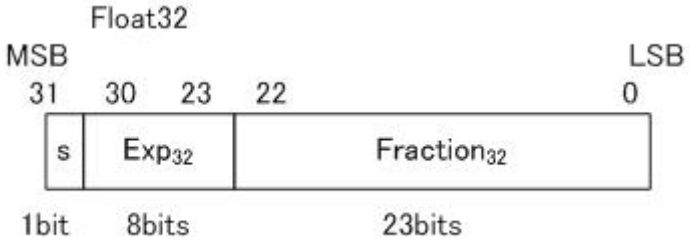


Figure 5 – Float32 type encode

Data of Float32 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 Float32 type shall be calculated by using the following formula:

$$\text{float32} = (-1)^s \times c_{32} \times 2^{q_{32}},$$

where

s : 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.

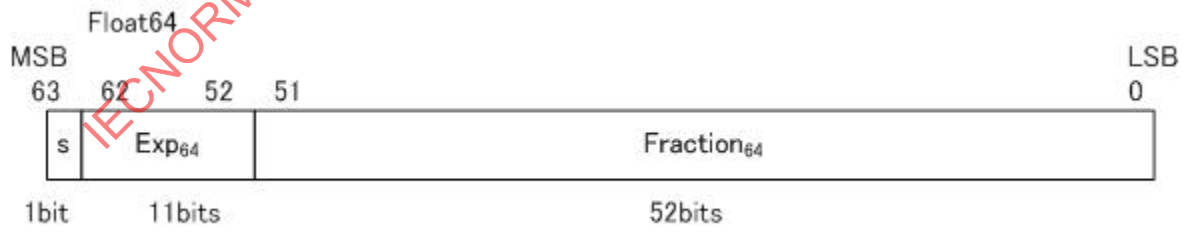


Figure 6 – Float64 type encode

Data of Float64 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 Float64 type shall be calculated by using the following formula:

$$\text{float64} = (-1)^s \times c_{64} \times 2^{q_{64}},$$

where

s : sign bit (0:plus / 1:minus),

$$c_{64} = \left(1 \times 2^{52} + Fraction_{64}\right) \times 2^{-52}$$
$$= 1 + \frac{b_{51}}{2} + \frac{b_{50}}{2^2} + \cdots + \frac{b_0}{2^{52}} \quad : \text{mantissa,}$$

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

If both $Fraction_{64}$ and $Exp64$ 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 may 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 according to ISO/IEC 8824-1:2008, 22.2.

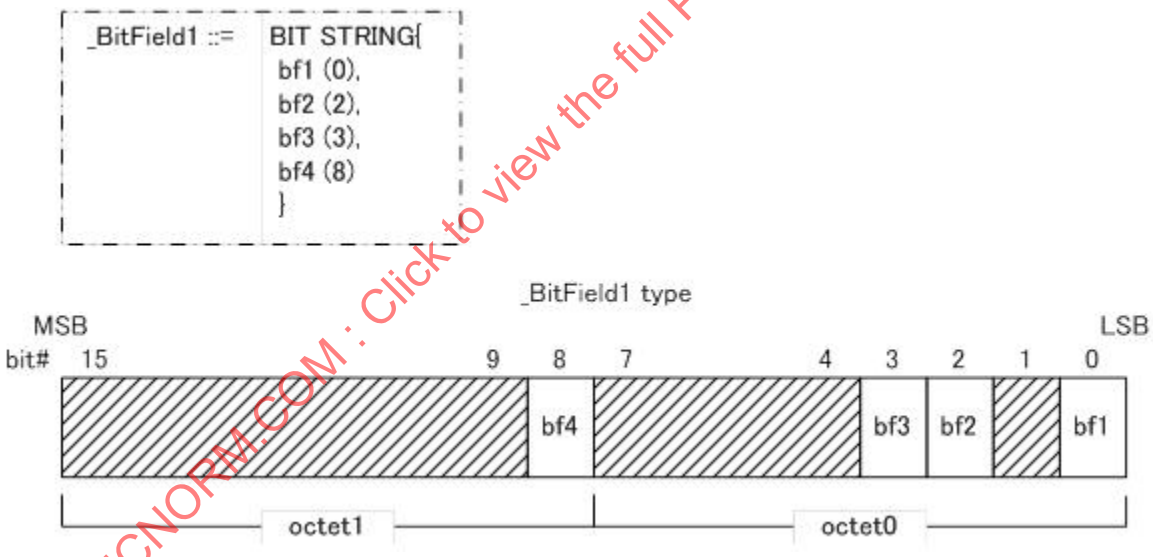


Figure 7 – Bit field definition example with named bits

A bit field may be defined by combining SEQUENCE type and BIT STRING type. In this case, the boundary of each field shall be defined according to the specified bit size and fill from the lower bit (see the example shown in Figure 8).

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.

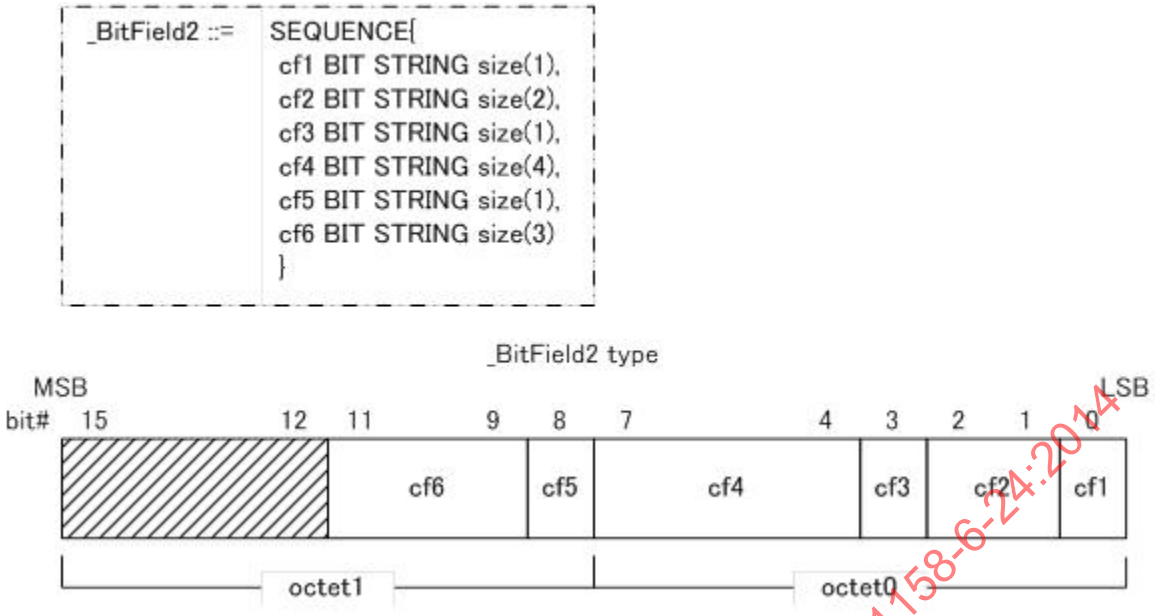


Figure 8 – Bit field definition example with field size

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.

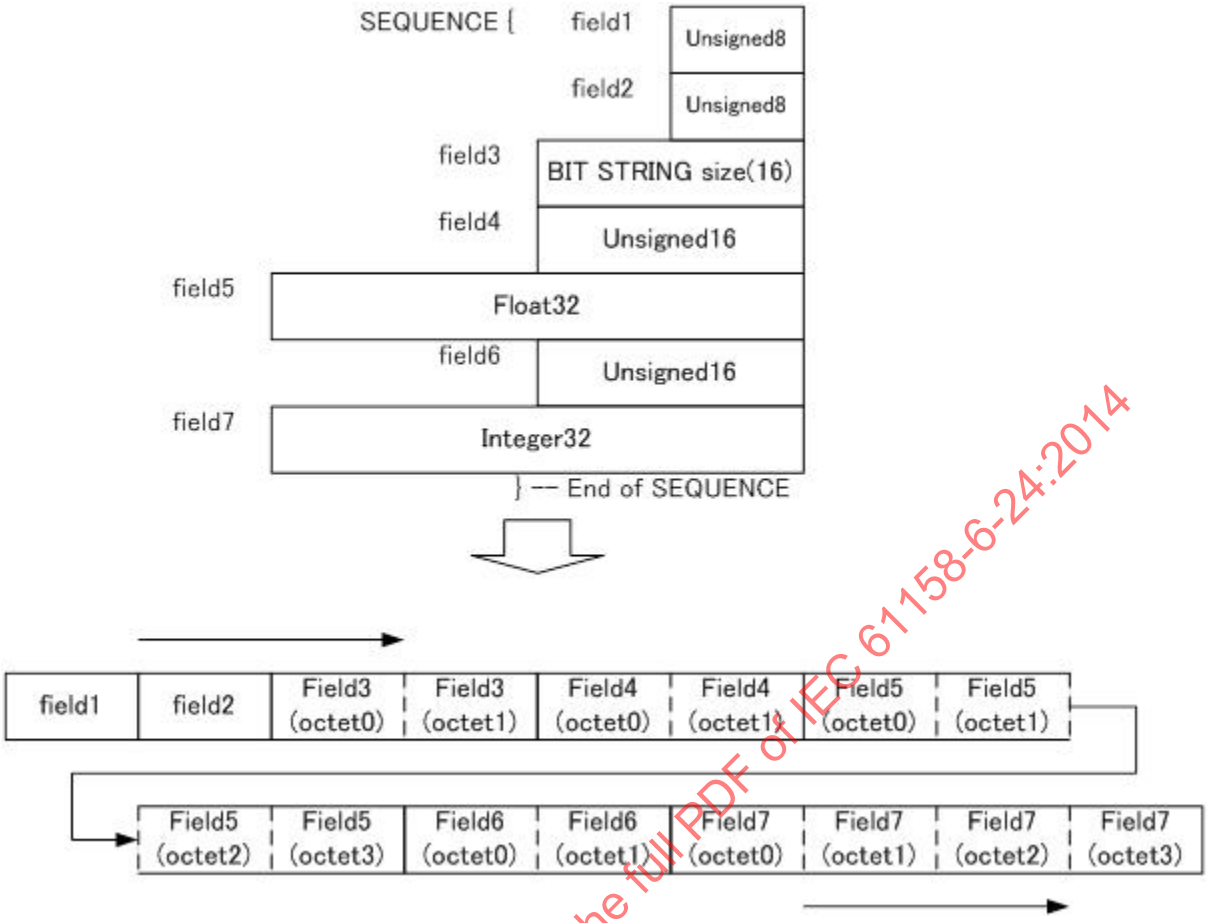


Figure 9 – SEQUENCE type encode

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 may depend on implementation and out of range of this standard. 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 characterised 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	Monitor	FDC PM-MN	FSPM
	MSG ASE	Requester	MSG PM-RQ	FSPM
		Responder	MSG PM-RS	
	AR ASE	FDCMonitor-AR	ARPM-FDCMN	ARPM with DMPM
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a

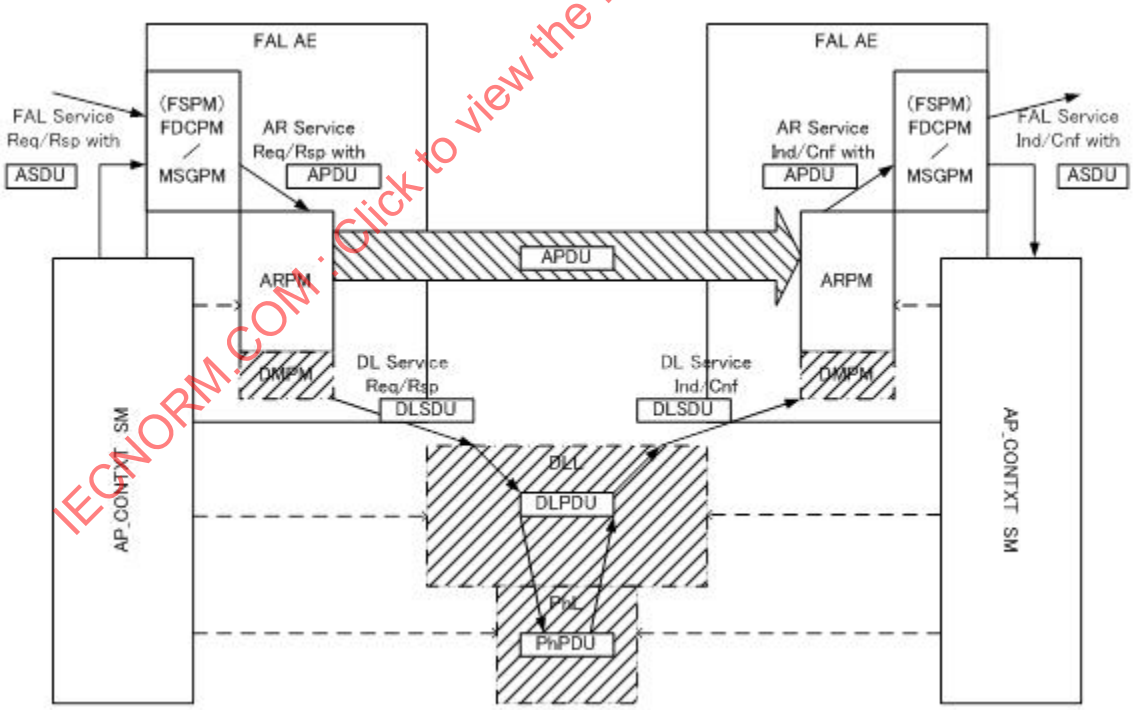


Figure 10 – Structure of FAL protocol state machines

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 may contain 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 may indirectly act 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.

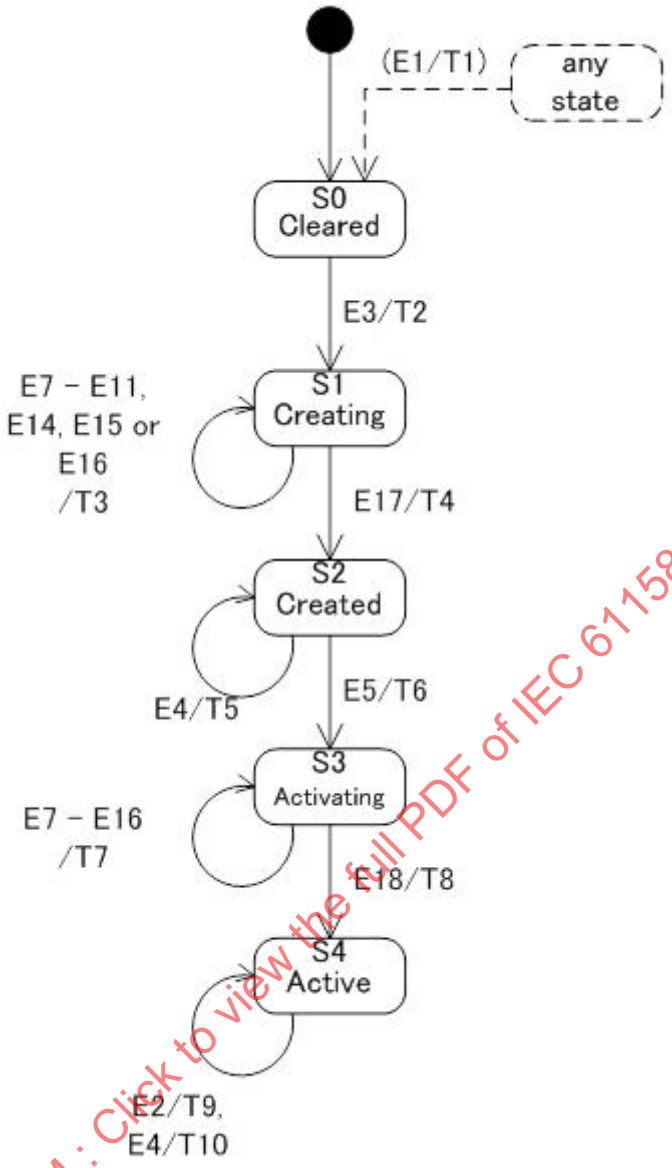


Figure 11 – Statechart diagram of APCSM

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

S#	State	Substate	Description
S4	Active	-	Steady state when all entities are activated and normal communication services are provided

7.3 Triggering events

Table 6 lists each trigger events 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 scope of this standard, 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
T2	S0:Cleared	E3: FSM-SetContext.req (CONTEXT-DATA) / /*-- APCSM starts initializing PhL	S1:Creating

T#	Source State	Event (arguments) [conditions] / action	Target State
		-- 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};	
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 takes 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) may be provided to the FAL user in the Type 24 FAL (refer to IEC 61158-5-24), and the FSPM may consist of two types of PM (FDC PM and MSG PM) corresponding to the ASEs.

8.2 Field Deice 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 may 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) in each communication cycle and should not accept any new command until the user completes the processing command.
	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 may wait the corresponding process completion response to the previous command. Until them, the user should not request to transfer any updated command. The FDC slave may receive 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) 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

Transmission mode (by DLL)	Communication state	Communication command type	Description
			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 shall start when the connection has 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 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.

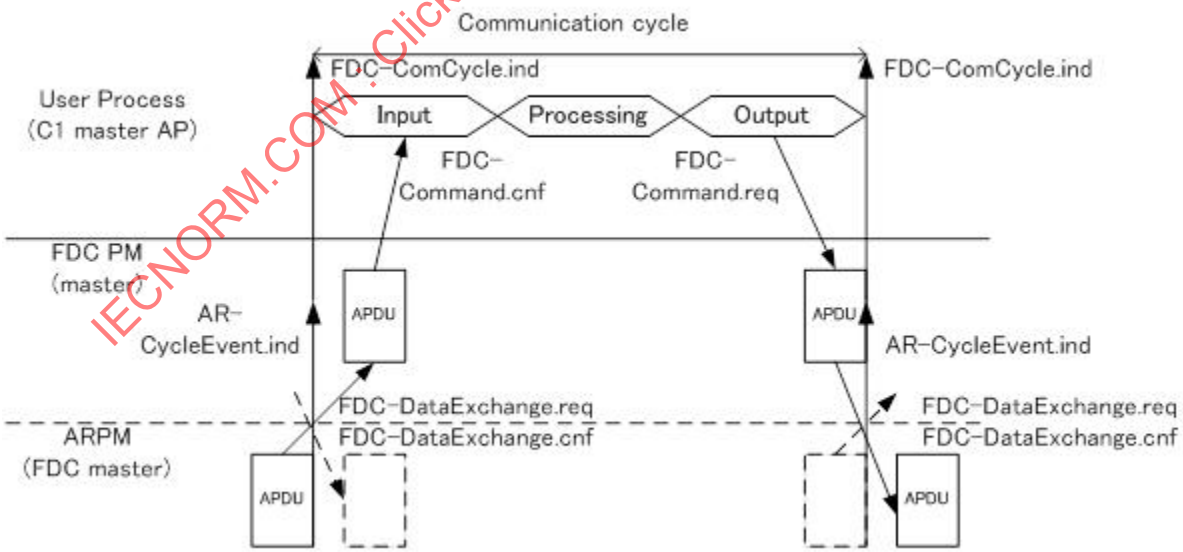


Figure 12 – Example communication cycle of FDC master AP

For example in the C1 master, the command output timing, the response input timing and the data processing timing may be 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 11,12 are shown as just informative examples.

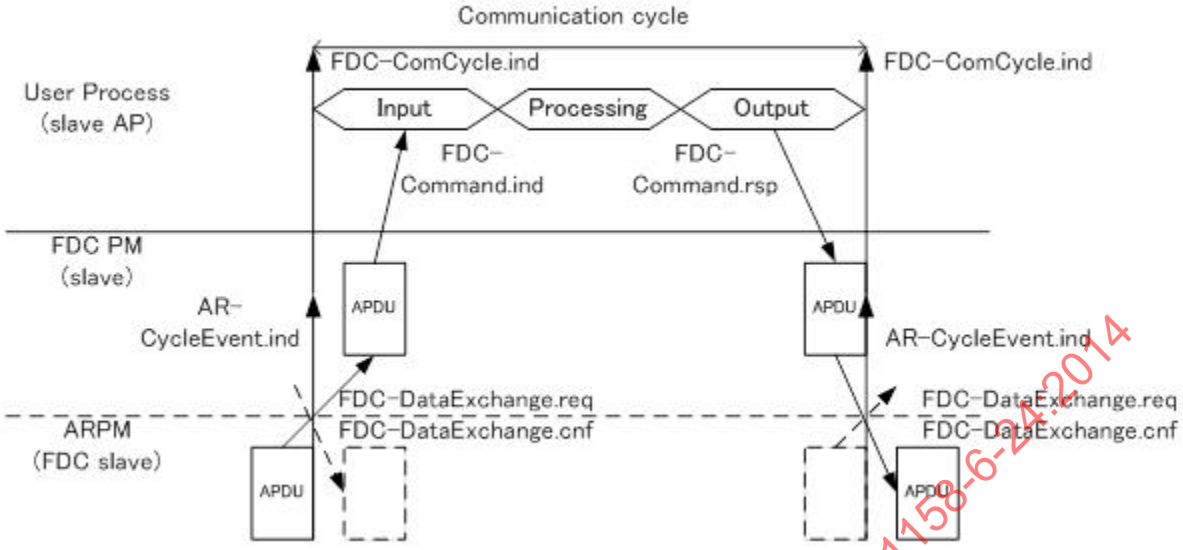


Figure 13 – Example communication cycle of FDC slave AP

In the same way for the slave AP, the command input timing, the response output timing and the data processing timing may be 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 to acknowledge that they are synchronized each other.

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

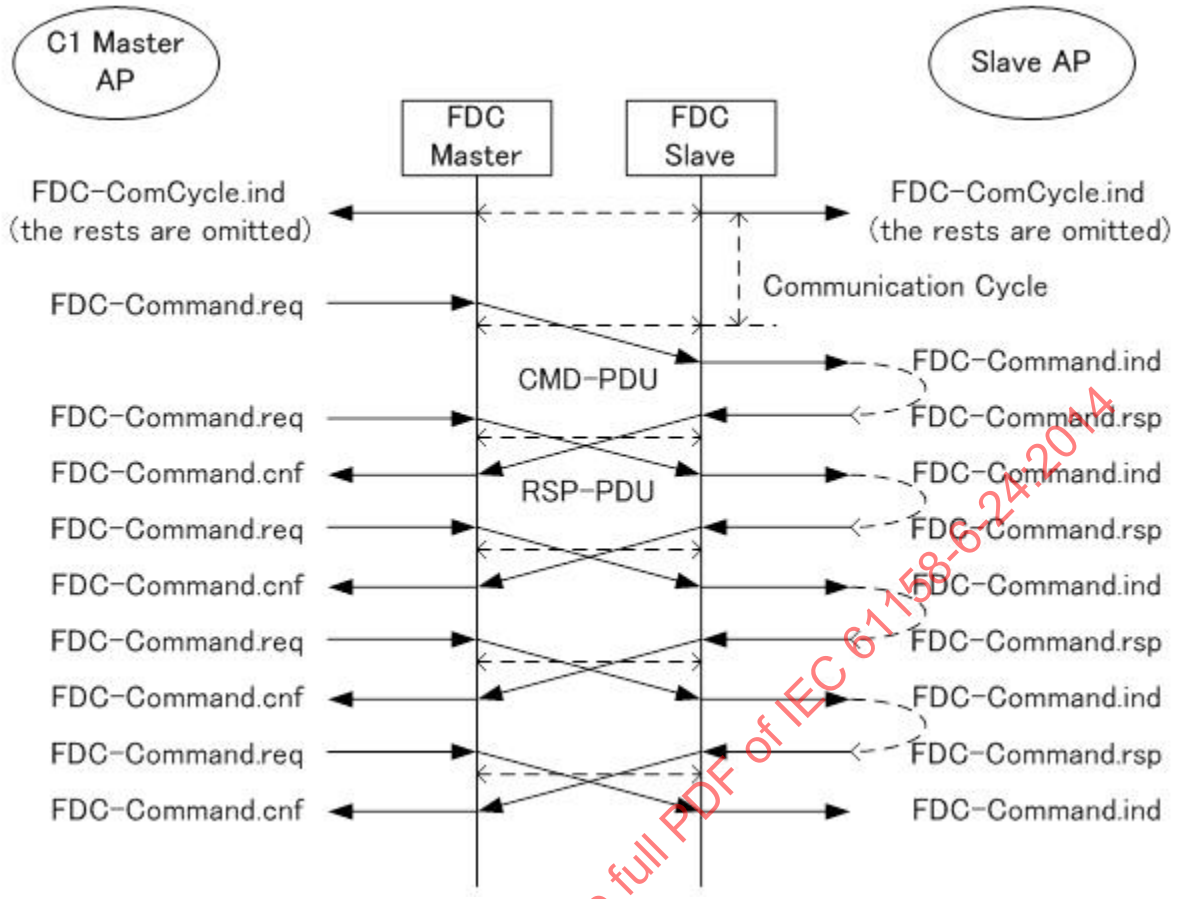


Figure 14 – Synchronous command communication in sync state

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”.

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 watching of the synchronized status through the WDT 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.

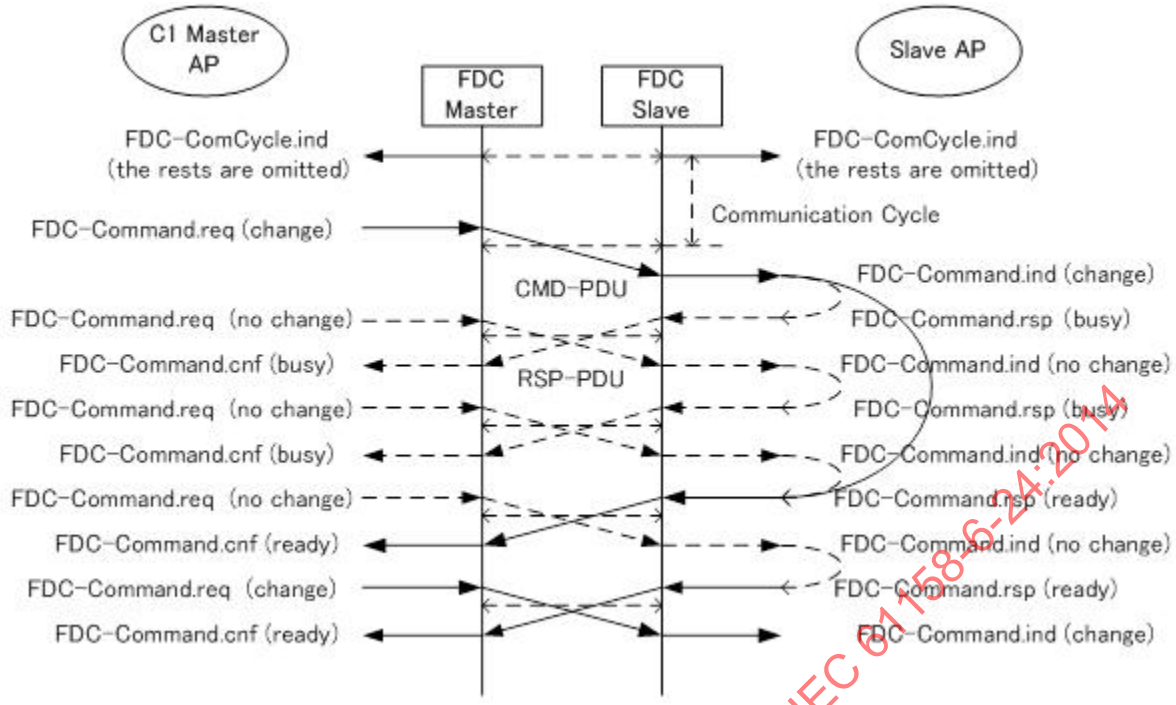


Figure 15 – Asynchronous command communication in sync state

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.

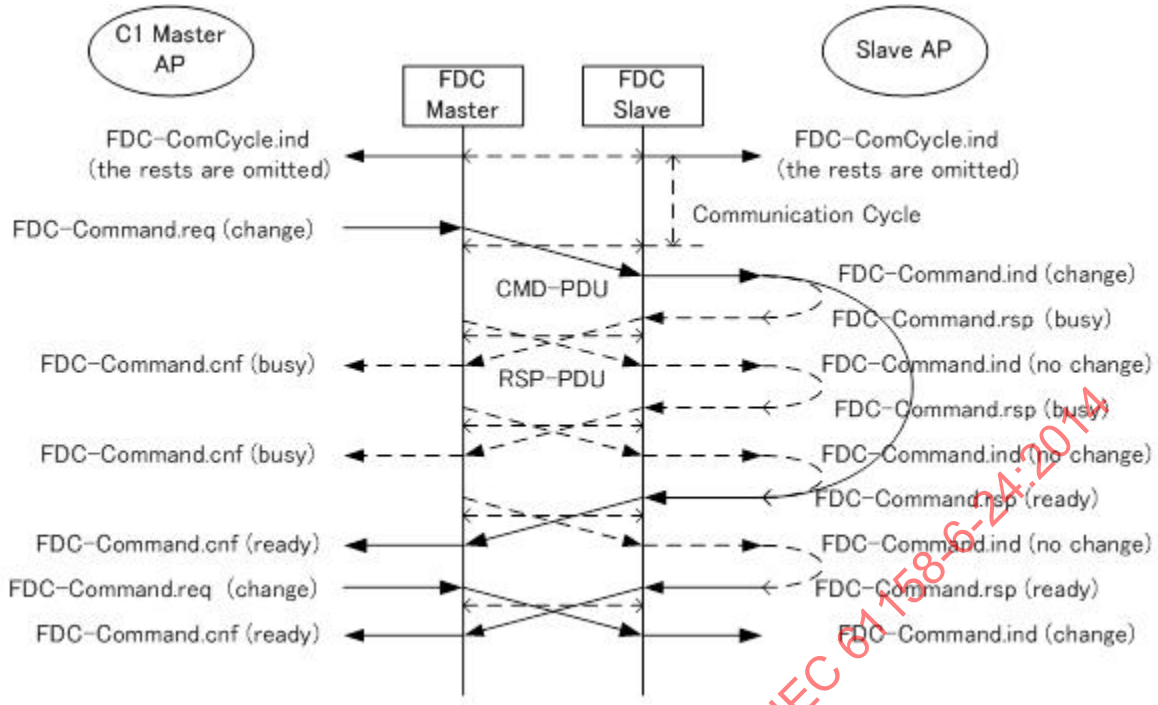


Figure 16 – Asynchronous command communication in async state

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 may only use asynchronous type commands. And the master AP may 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.

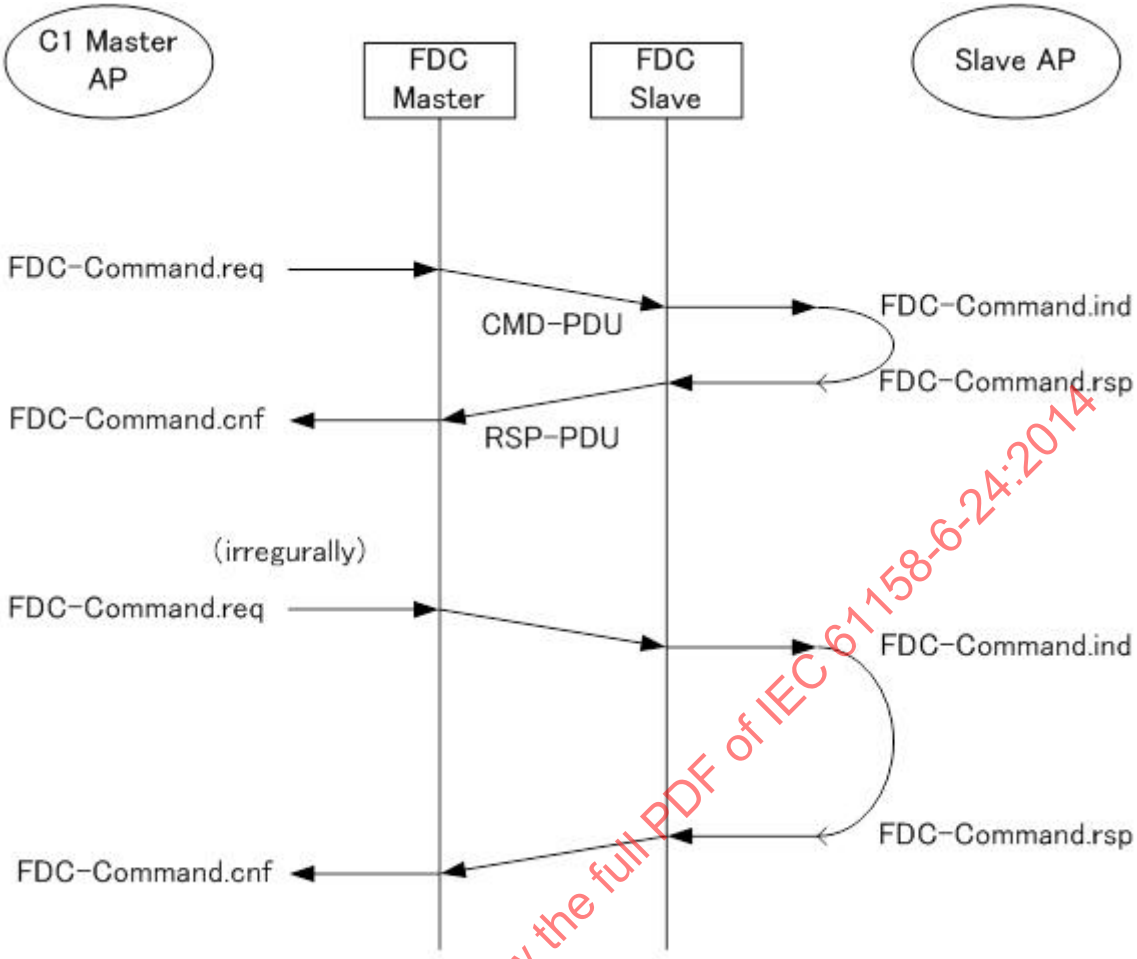


Figure 17 – Event-driven communication

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.

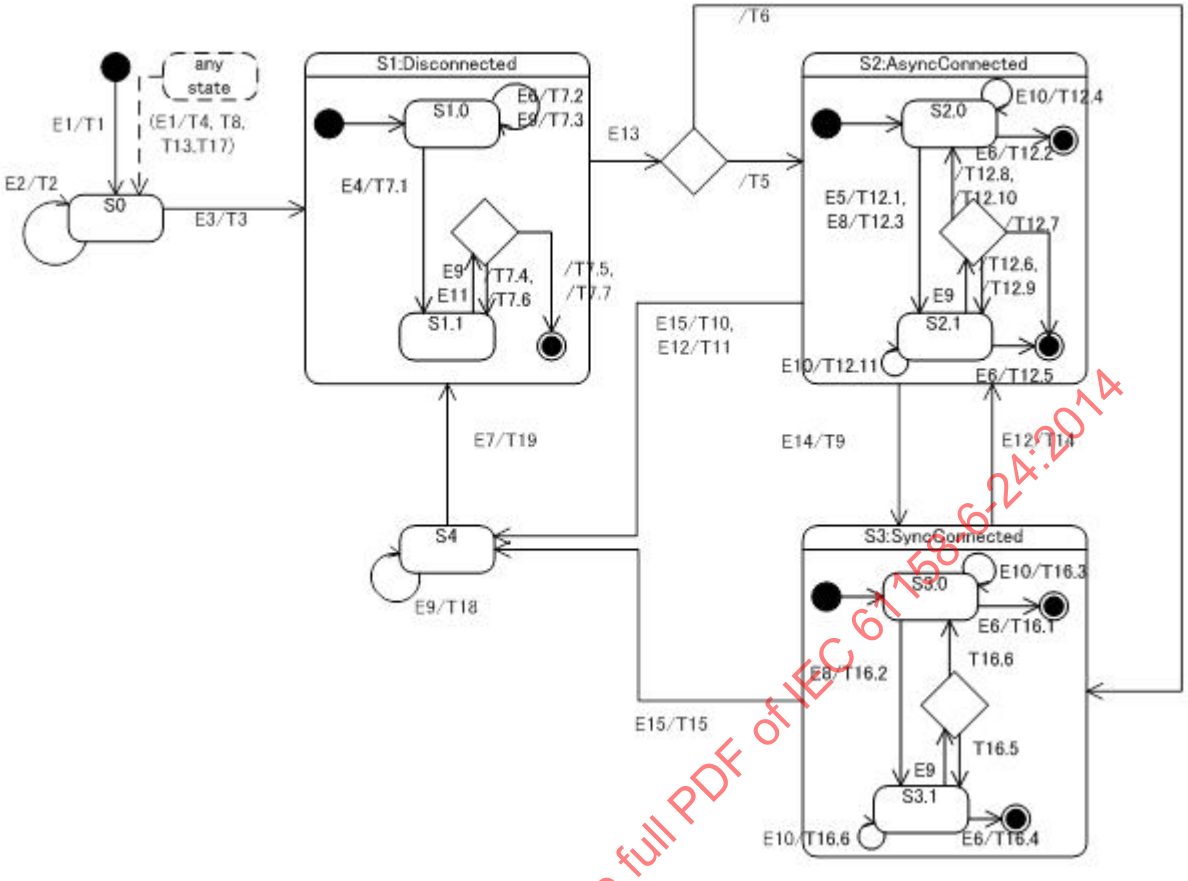


Figure 18 – Statechart diagram of FDCPM-M

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 may be 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 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 a request to transfer a CMD-PDU from a FAL user
S3.1		WaitCMDComplete	Substate when the PM is waiting a command completion response from the correspondent peer 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).

8.2.4.2 Triggering events

Table 10 lists each trigger events 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		

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
		(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
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 ;	S2:AsyncConnected

T#	Source State	Event (arguments) [conditions] / action	Target State
		DevProfileType = rsdu.rspBody.profile_type;	
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
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)	S1.0: Idle

T#	Source State	Event (arguments) [conditions] / action	Target state
		AR-SendCommand.req (csdu); else /*-- no-operation --*/;	
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: WaitCMDComplete	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: WaitCMDComplete	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: 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; AR-SendCommand.req (csdu); /* Comment: TransMode is EventDriven. */	S1.1: WaitCMDComplete
T7.7	S1.1: WaitCMDComplete	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
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: WaitCMDComplete	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: WaitCMDComplete	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: WaitCMDComplete	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);	Exit to the main machine

T#	Source State	Event (arguments) [conditions] / action	Target state
		FDC-SyncSet.cnf(+); /*-- signal E14:EventSyncSetRSP to the main machine --*/;	
T12.8	S2.1: WaitCMDComplete	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

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.

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.

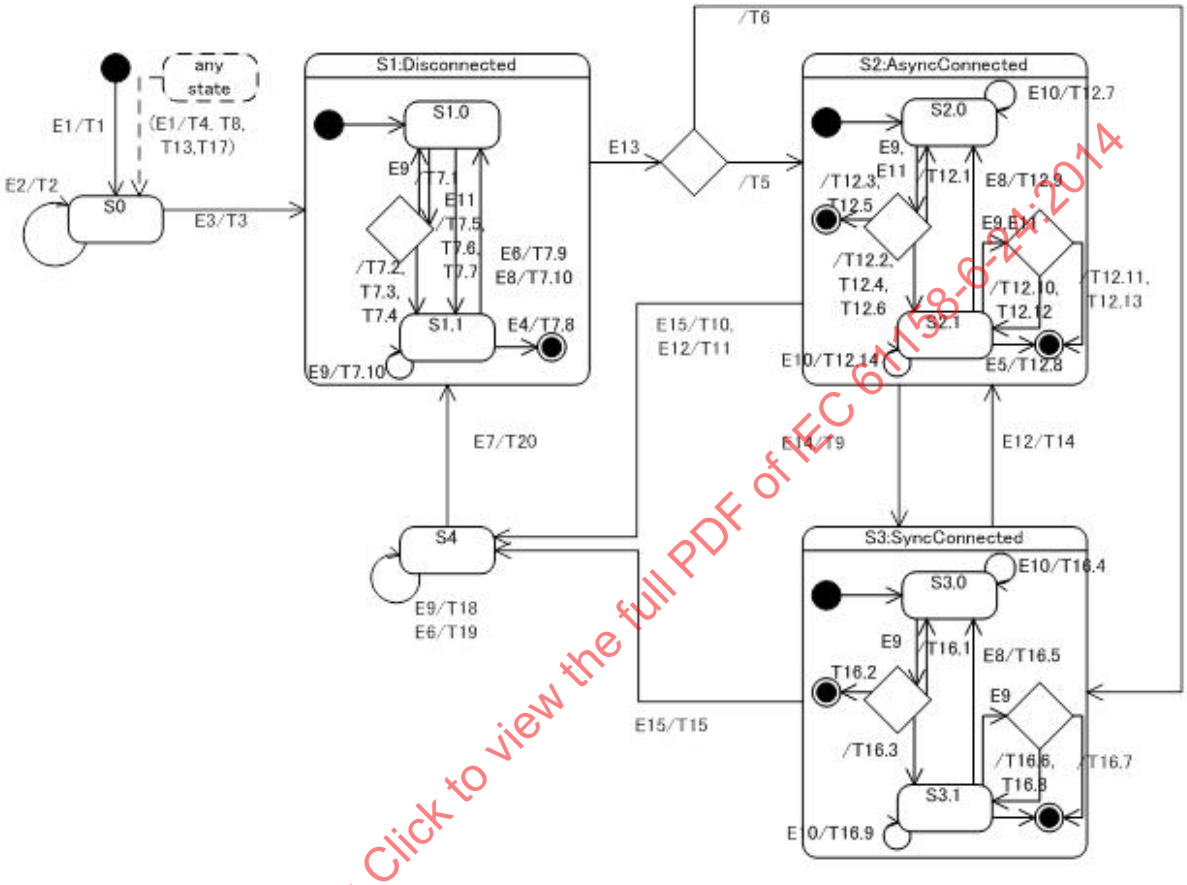


Figure 19 – Statechart diagram of FDCPM-S

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.requet 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 may be 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.

S#	State	Substate	Description
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.
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 events 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-	

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
			SDU	
E6	FDC-Disconnect.rsp	FAL user (Slave AP)	ServiceStatus, ProgressStatus, RSP-SDU	
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
T5	S1:Disconnected	E13:EventConnectRSP (status, rsdu)	S2:AsyncConnected

T#	Source State	Event (arguments) [conditions] / action	Target State
		[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;	
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 [1)] / /*-- 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 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: WaitRSPComplete
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: WaitRSPComplete
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: WaitRSPComplete
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: WaitRSPComplete
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: WaitRSPComplete
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: WaitRSPComplete
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: WaitRSPComplete
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-DataExchnage.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: WaitRSPComplete
T12.3	S2.0:Idle	E9:FDC-DataExchnage.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU;	exit from the submachine

T#	Source State	Event (arguments) [conditions] / action	Target state
		FDC-DataExchange.cnf (rsdu); FDC-Disconnect.ind (csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	
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-DataExchnage.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
T12.12	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: WaitRSPComplete
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: WaitRSPComplete

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-DataExchnage.req (csdu) [(csdu != LastCMD-SDU) && (csdu.cmd == disconnect)] / LastCMD-SDU = csdu; rsdu = LastRSP-SDU; FDC-DataExchange.cnf (rsdu);	exit from the submachine

T#	Source State	Event (arguments) [conditions] / action	Target state
		FDC-Disconnect.ind (csdu); /*-- signal E15:EventDisconnectCMD to the main machine --*/;	

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: WaitRSPComplete
			or
			Exit to the main machine, when E12 has been issued.

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-DataExchnage.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.

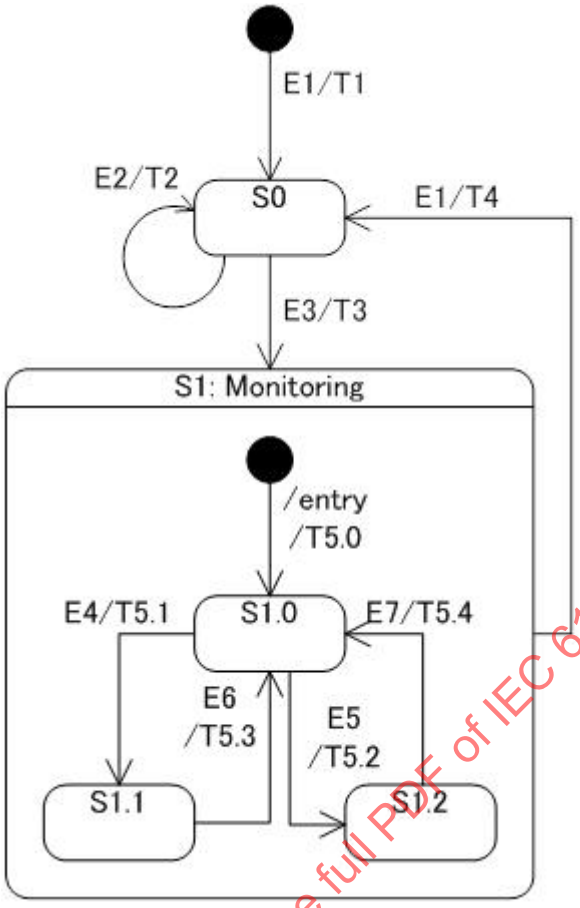


Figure 20 – Statechart diagram of FDCPM-MN

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.requet from a FSM ASE.
			Entry/ Initial values are set in Attributes of the Monitor object.
S1	Monitoring		An 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 events 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:WaitCMDData	E6:AR-GetCMD.cnf (+) (CMD_ASDU) / FDC-GetCMD.cnf (+) (CMD_ASDU)	S1.0:Idle
T5.4	S1.2:WaitRSPData	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 standard. 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 scope of this standard.

The following errors are defined:

a) communication error: receipt status such as no DL-PDU receipt and FCS error detected in the lower layer;

b) 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;

c) illegal cycle timing: abnormal value (beyond the jitter allowance) in the transmission cycle detected by the slave;

d) response timeout: response time-out detected by the master;

e) 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-DataExhange 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. And the RSP-PDU shall be edited in the way how the cmd_stat field or the status field should show the error as one of alarm factors.

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 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. And the RSP-PDU shall be edited in

the way how the cmd_stat field or the status field should show the error as one of alarm factors.

8.2.7.4 Illegal cycle timing

The “illegal cycle timing” is a kind of errors that shows that an FDC slave detects an abnormal cycle jitter by measuring the transmission cycle period. At the 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 an FDC slave PM detects the error in the synchronous state, it shall make the state transition to an asynchronous communication state, not to accept synchronous commands after that. And the RSP-PDU shall be edited in the way how the cmd_stat field or the status field should show the error as one of alarm factors.

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

8.2.7.5 Response timeout

The "response timeout" is a kind of errors 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. For more details of the recovery process for both the master and the slave, however, is dependent on the implementation.

8.2.7.6 Illegal Sync command

The "illegal sync command" is a kind of errors 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 RSP-PDU shall be edited in the way how the cmd field in the CMD-PDU echoes back with rcmd field in the RSP-PDU and the cmd_stat field or the status field should show the error as one of alarm factors.

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 troubles 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 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 may 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 Figure 21.

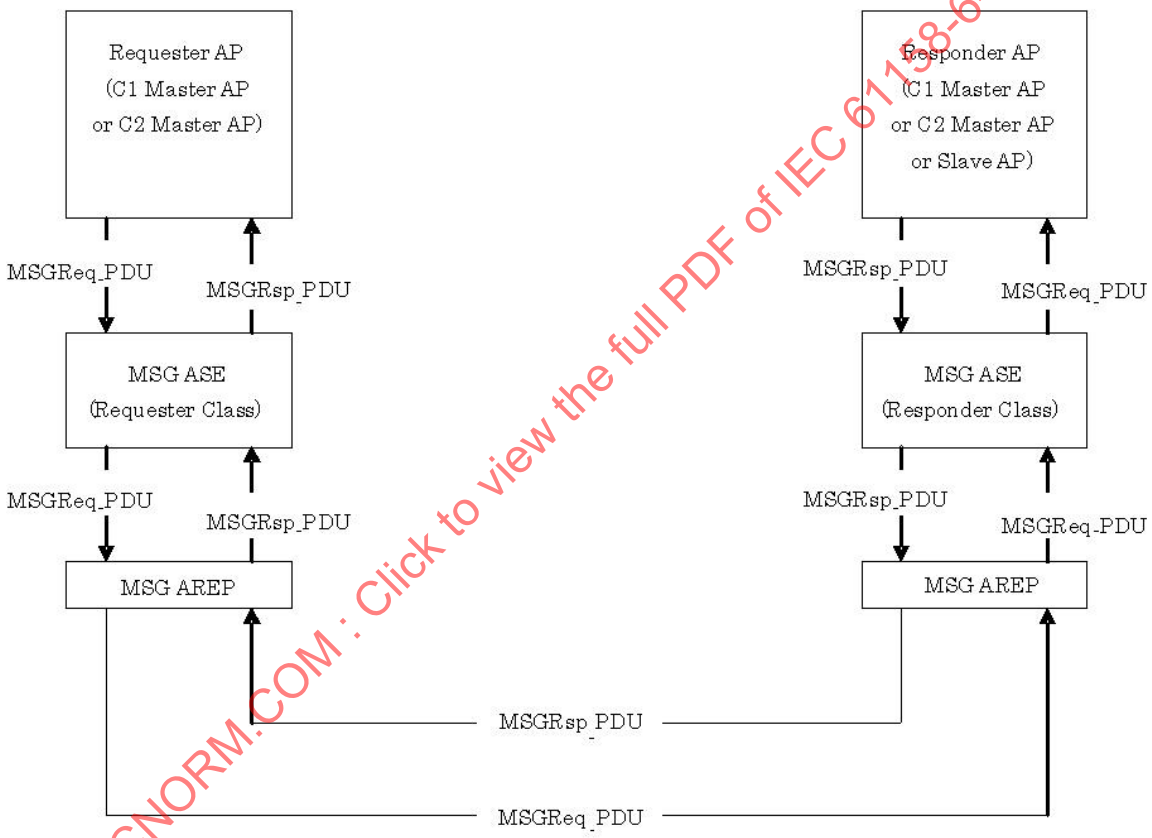


Figure 21 – PDU transmission flow for user message

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 Figure 22.

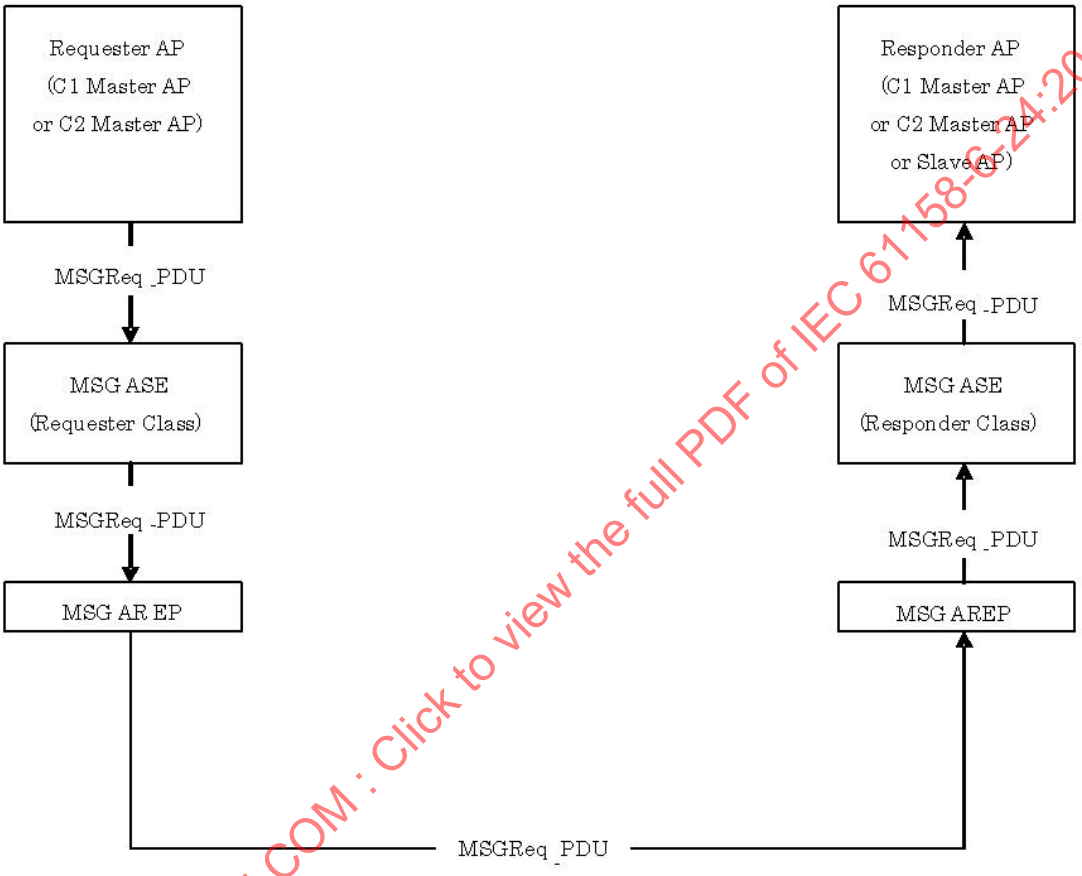


Figure 22 – PDU transmission flow for one-way message

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.

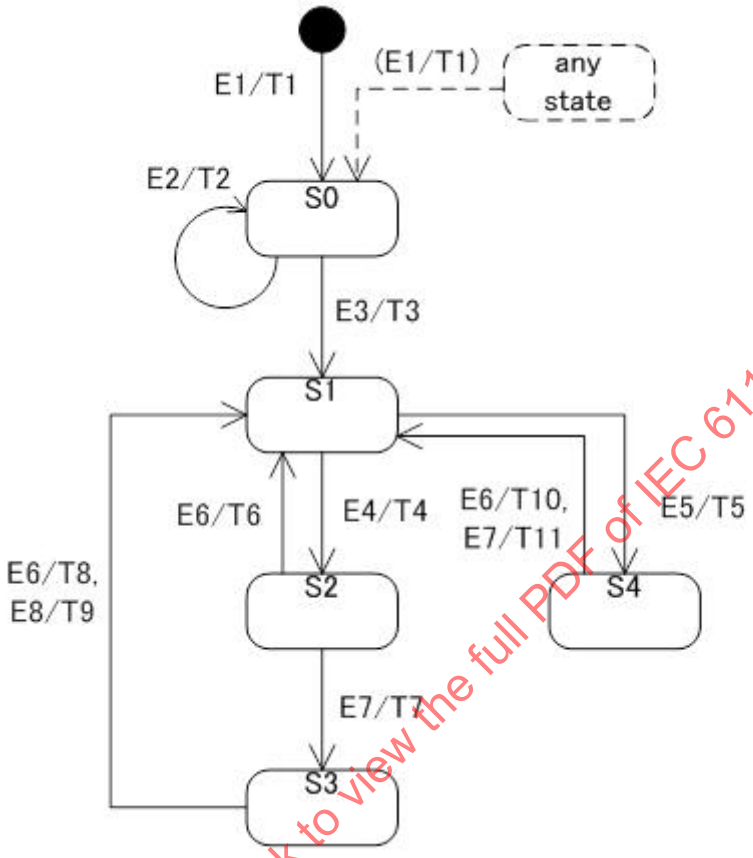


Figure 23 – Statechart diagram of MSGPM-RQ

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.requet 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 events 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
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.

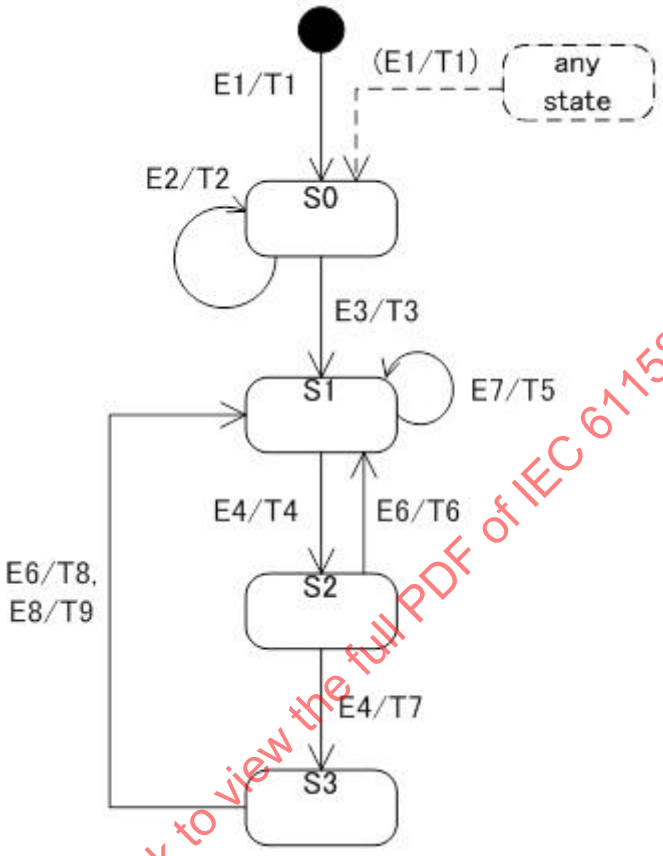


Figure 24 – Statechart diagram of MSGPM-RS

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 events 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)

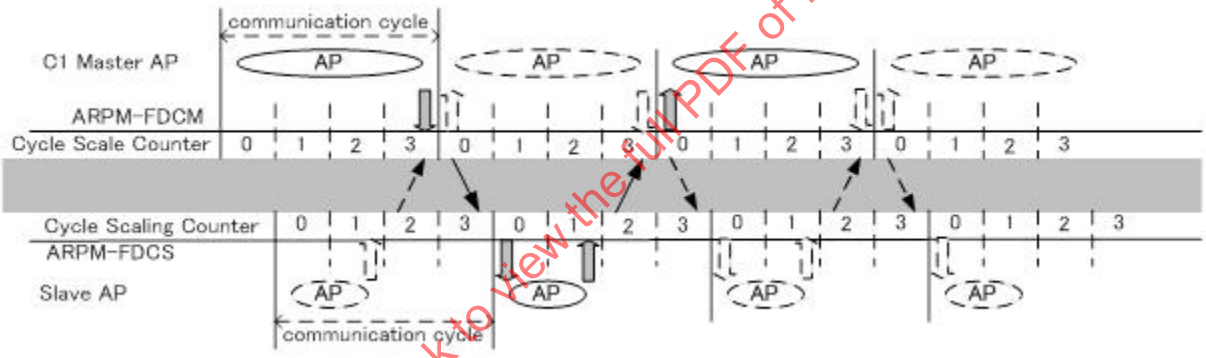
9.1 General

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 $T_{COMCYC} = T_{CYC} \times 4$,

where

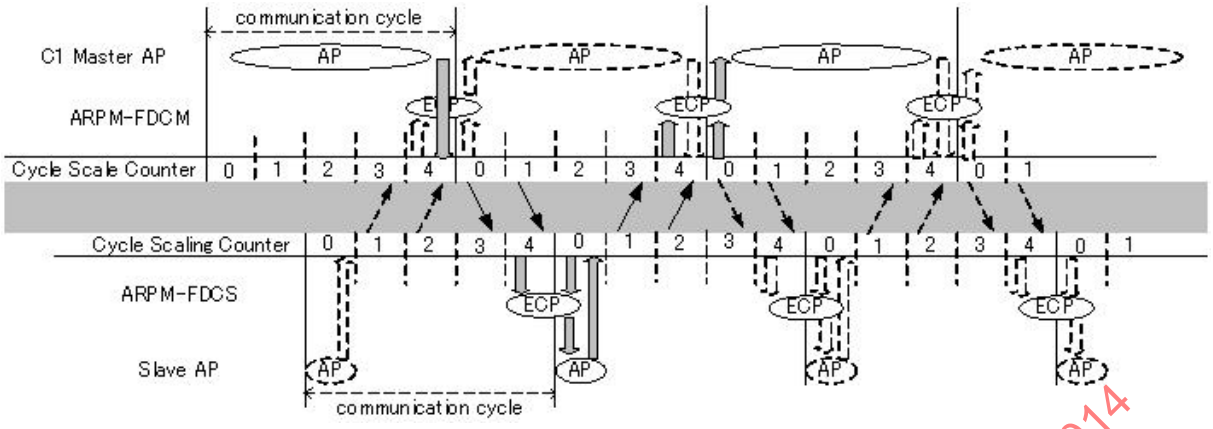
T_{COMCYC}: communication cycle;

T_{CYC}: 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 $T_{COMCYC} \geq T_{CYC} \times 2$, where T_{COMCYC} is a communication cycle and T_{CYC} is a transmission cycle.



NOTE The figure shows the case of $T_{COMCYC} = T_{CYC} \times 5$,

where

T_{COMCYC}: communication cycle;

T_{CYC}: transmission cycle;

ECP: Error-detection and PDU-comparison process.

Figure 26 – Example of dual transfer process

9.2.2 ARPM for FDC Master (ARPM-FDCM)

9.2.2.1 State descriptions

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

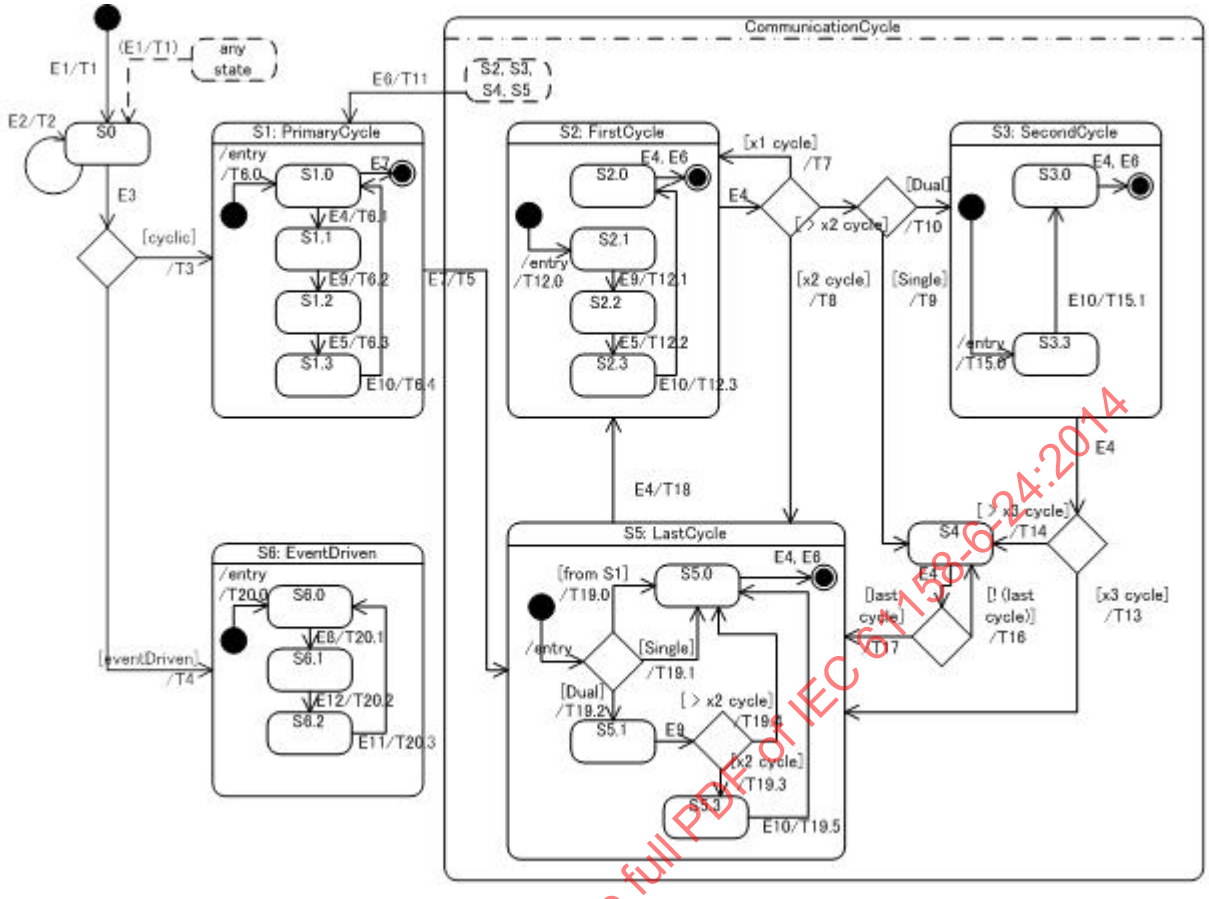


Figure 27 – Statechart diagram of ARPM-FDCM

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 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
S5.3		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, if the communication cycle is twice primary cycle

S#	State	Substate	Description
S6	EventDriven	-	Corresponding state to the event-driven mode in DLL A CMD-PDU is not cyclically transmitted but in case of necessity it's done and after that, sequentially RSP-PDU is received using DL services provided in the event-driven mode. entry/ Transition into the sub state: S6.0.
S6.0		WaitRequestCMD	Substate when the PM is waiting for AR-SendCommand.req from FDC ASE
S6.1		SendingRequestCMD	Substate when the PM is waiting for completion of DL_SDN.req
S6.2		WaitConformCMD	Substate when the PM is waiting for the reception of DL_SDN.ind from DLL as a RSP-PDU

9.2.2.2 Triggering events

Table 28 lists each trigger events of the ARPM-FDCM.

Table 28 – Trigger event descriptions of ARPM-FDCM

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		
E2	AR-Open.req	FSM ASE	DLSAPID, FDCSAPID	
E3	AR-Enable.req	FSM ASE	TransmissionMode	
E4	EVM-SyncEvent.ind	EVM ASE		
E5	FDC-DataExchange.cnf	FDC ASE (Master)	CMD-SDU	
E6	AR-ResetCycle.req	FDC ASE (Master)		
E7	AR-StartComCycle.req	FDC ASE (Master)	ComTime, DTMode, CMDForm	
E8	AR-SendCommand.req	FDC ASE (Master)	CMD-SDU	
E9	DL-READ-DATA.cnf	DLL	Result, DLSDU	
E10	DL-WRITE-DATA.cnf	DLL	Result	
E11	DL-SDN.ind	DLL	SAP_ID, Node_ID, Length, DLSDU	
E12	DL-SDN.cnf	DLL	Result	
E13	Error detected	FDC ASE (Master)		See 8.2.7

9.2.2.3 Action descriptions at state transitions

Table 29 describes state transitions of the main SM of ARPM-FDCM. Moreover, Table 30 describes state transitions of the submachine of ARPM-FDCM.

Table 29 – Transitions of main SM of ARPM-FDCM

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1:AR-Reset.req / /*-- Clearing all attributes of the FDC-Master AR --*/;	S0:Disabled
T2	S0:Disabled	E2:AR-Open.req / /*-- Initializing the FDC-Master AR --*/; [a]	S0:Disabled
T3	S0:Disabled	E3:AR-Enable.req (comMode) [comMode == “cyclic”] / /*-- no-operation --*/;	S1:PrimaryCycle
T4	S0:Disabled	E3:AR-Enable.req (comMode) [comMode == “eventDriven”] / /*-- no-operation --*/;	S6:EventDriven
T5	S1:PrimaryCycle	E7:AR-StartComCycle / /*-- no-operation --*/;	S5:LastCycle
T6	S1:PrimaryCycle	The other events / /*-- state transition in the submachine --*/;	S1:PrimaryCycle
T7	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 1] / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Master objects;	S2:FirstCycle
T8	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 2] / CycleScaleCounter++;	S5:LastCycle
T9	S2:FirstCycle	E4:EVM-SyncEvent.ind [(ComTime > 2) && (DTMode == “Single”)] / CycleScaleCounter++;	S4:MiddleCycle
T10	S2:FirstCycle	E4:EVM-SyncEvent.ind [(ComTime > 2) && (DTMode == “Dual”)] / CycleScaleCounter++;	S3:SecondCycle
T11	S2, S3, S4, or S5	E6:AR-ResetCycle.req / /*-- no-operation --*/;	S1:PrimaryCycle
T12	S2:FirstCycle	The other events / /*-- state transition in the submachine --*/;	S2:FirstCycle
T13	S3:SecondCycle	E4:EVM-SyncEvent.ind [ComTime == 3] / CycleScaleCounter++;	S5:LastCycle
T14	S3:SecondCycle	E4:EVM-SyncEvent.ind [ComTime > 3] / CycleScaleCounter++;	S4:MiddleCycle
T15	S3:SecondCycle	The other events / /*-- state transition in the submachine --*/;	S3:SecondCycle
T16	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter < (ComTime-2)] / CycleScaleCounter++;	S4:MiddleCycle
T17	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter == (ComTime-2)] / CycleScaleCounter++;	S5:LastCycle
T18	S5:LastCycle	E4:EVM-SyncEvent.ind / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Master objects;	S2:FirstCycle
T19	S5:LastCycle	The other events / /*-- state transition in the submachine --*/;	S5:LastCycle
T20	S6:EventDriven	Any events except E1 / /*-- state transition in the submachine --*/;	S6:EventDriven

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

Table 30 – Transitions of submachine of ARPM-FDCM

T#	Source State	Event (arguments) [conditions] / action	Target State
T6.0 (T3)	(S0)	/entry / /*-- no-operation --*/;	S1.0:Idle
T6.1	S1.0:Idle	E4:EVM-SyncEvent.ind / DL_GET_DATA.req ;	S1.1:WaitDL_GET_DATAcnf
T6.2	S1.1:WaitDL_GET_DATAcnf	E9:DL_GET_DATA.cnf (rsdu) / FDC-DataExchange.req (rsdu);	S1.2:WaitFDC-DataExchangecnf
T6.3	S1.2:WaitFDC-DataExchangecnf	E5:FDC-DataExchange.cnf (csdu) / DL_SET_DATA.req (csdu);	S1.3:WaitDL_SET_DATAcnf
T6.4	S1.3:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S1.0:Idle
T12.0	(S2 or S5)	/entry	S.2.1:WaitDL_GET

T#	Source State	Event (arguments) [conditions] / action	Target State
(T7, T18)		/ DL_GET_DATA.req ;	_DATAcnf
T12.1	S2.1:WaitDL_GET_DATAcnf	E9: DL_GET_DATA.cnf (rsdu) / if (DTMode == “Dual”) { /*-- Error detection and PDU comparison process -- with lastRsdu and rsdu --*/} else {/*-- no-operation --*/;} FDC-DataExchange.req (rsdu);	S.2.2: WaitFDC-DataExchangecnf
T12.2	S2.2: WaitFDC-DataExchangecnf	E5:FDC-DataExchange.cnf (csdu) / DL_SET_DATA.req (csdu); if (DTMode == “Dual”) { lastCsdu = csdu;}	S2.3: :WaitDL_SET_DATAcnf
T12.3	S2.3:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S.2.0:Idle
T15.0 (T10)	(S2:FirstCycle)	/entry / DL_SET_DATA.req (csdu) ;	S3.3:WaitDL_SET_DATAcnf
T15.1	S3.3:WaitDL_SET_DATAcnf	E10: DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S3.0:Idle
T19.0 (T5)	(S1:PrimaryCycle)	/entry / /*-- no-operation --*/;	S5.0:Idle
T19.1	(S2, S3, or S4)	/entry [DTMode == “Single”] / /*-- no-operation --*/;	S5.0:Idle
T19.2	(S2, S3, or S4)	/entry: [DTMode == “Dual”] / DL_GET_DATA.req ;	S5.1:WaitDL_GET_DATAcnf
T19.3	S5.1:WaitDL_GET_DATAcnf	E9: DL_GET_DATA.cnf (rsdu) [ComTime == 2] / lastRsdu = rsdu; DL_SET_DATA.req (lastCsdu);	S5.3:WaitDL_SET_DATAcnf
T19.4	S5.1:WaitDL_GET_DATAcnf	E9: DL_GET_DATA.cnf (rsdu) [ComTime > 2] / lastRsdu = rsdu;	S5.0:Idle
T19.5	S5.3:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S5.0:Idle
T20.0 (T4)	(S0)	/entry: / /*-- no-operation --*/;	S6.0:WaitRequestCMD
T20.1	S6.0:WaitRequestCMD	E8: AR-SendCommand.req (csdu) / DL_SDN.req (SAP_ID, Node_ID, Length, DLSDU) ;	S6.1:SendingRequestCMD
T20.2	S6.1:SendingRequestCMD	E12: DL_SDN.cnf / /*-- no-operation --*/;	S6.2:WaitConformCMD
T20.3	S6.2:WaitConformCMD	E11: DL_SDN.ind (SAP_ID, Node_ID, Length, DLSDU) / AR-SendCommand.cnf (ServiceStatus, rsdu);	S6.0:WaitRequestCMD
T20.4	S6.2:WaitConformCMD	E13 Communication time-out / /*-- Communication alarm processing and resume processing --*/;	S6.0:WaitRequestCMD
NOTE “lastCsdu” and “lastRsdu” are local variables.			

9.2.3 ARPM for FDC Slave (ARPM-FDCS)

9.2.3.1 State descriptions

Figure 28 shows the ARPM-FDCS statechart diagram, and Table 31 describes each state of the ARPM-FDCS.

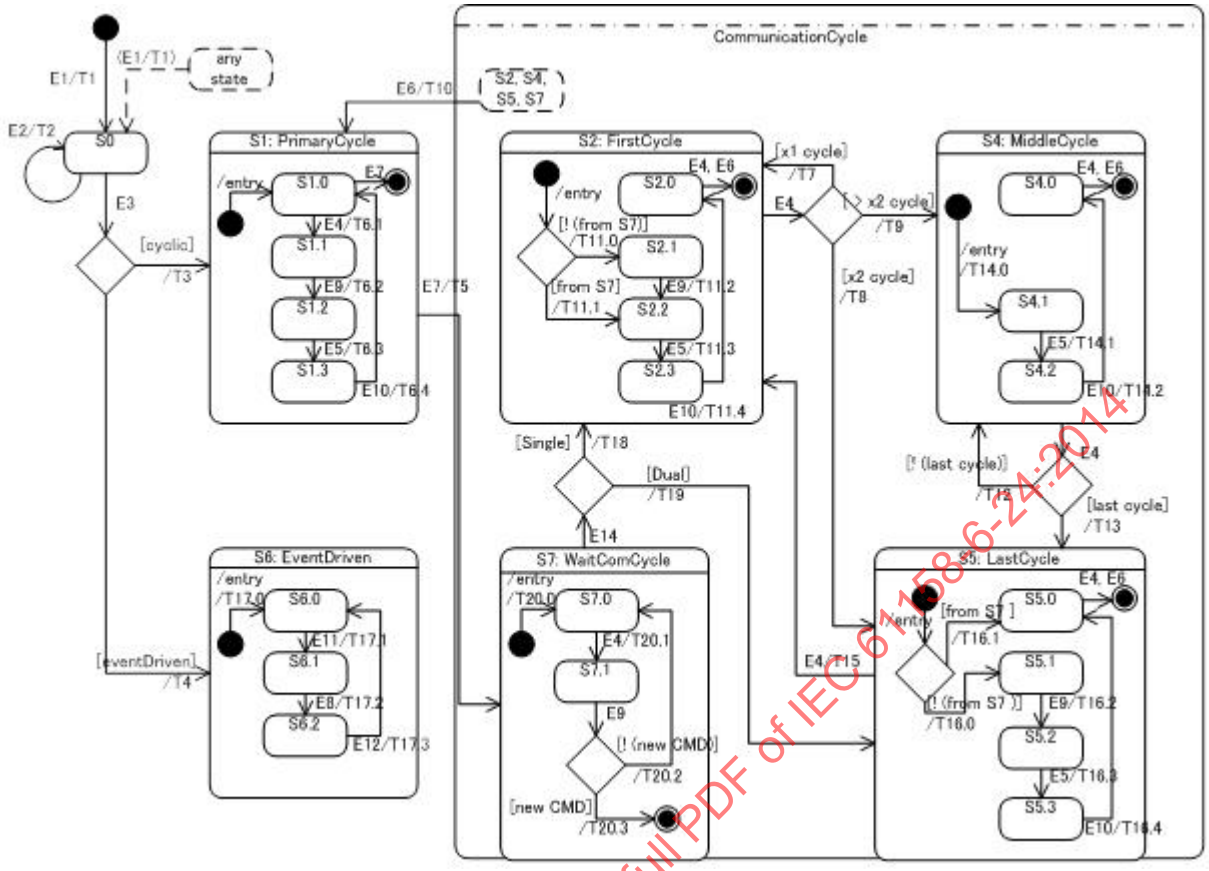


Figure 28 – Statechart diagram of ARPM-FDCS

Table 31 – State descriptions of ARPM-FDCS

S#	State	Substate	Description
S0	Disabled		State when the FDC Slave AR Class has just 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 FDC Slave 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 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 where 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-DataExchange	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-FDCS exchanges PDUs between the FDC ASE and DLL If the communication cycle is same as the primary cycle, the PM remains in this state while the FDC PM is in connected. entry [the source state != S7] / Transition into the substate: S2.1. entry [the source state == S7] / Transition into the substate: S2.2.
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_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
S2.2		WaitFDC-DataExchange	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_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 in each middle primary cycle of the communication cycle the ARPM-FDCS relay a RSP-PDU from the FDC ASE to DLL
S4.0		Idle	Substate when the PM is waiting for the start signal based on the network clock for the next primary cycle
S4.1		WaitFDC-DataExchange	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S4.2		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
S5	LastCycle		State when a connection in FDC ASE is established, and in the last primary cycle of the communication cycle the ARPM-FDCS relay a RSP-PDU from the FDC ASE to DLL In the dual transfer mode, the PM receives a CMD-PDU from the master and stores it into the internal buffer before the regular PDU exchange in the first cycle. entry [(the source state != S7) && dual transfer mode] / Transition into the sub state: S5.1. entry [(the source state == S7) || single transfer mode] / Transition into the sub state: S5.2.
S5.0		Idle	Substate when the PM is waiting for the start signal 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 if in the dual transfer mode
S5.2		WaitFDC-DataExchange	Substate when the PM is waiting for the associated FDC-DataExchange.cnf after requesting the DataExchange.req to the related FDC-SAPID
S5.3		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
S6	EventDriven		Corresponding state to the event-driven mode in DLL A CMD-PDU is not cyclically received but as in case of necessity, it's done and after that, sequentially a RSP-PDU is transmitted using DL services in the event-driven mode. entry/ Transition into the sub state: S6.0.
S6.0		WaitCMDIndication	Substate when the PM is waiting for the DL_SDN.ind from DLL, which originates with the peer master AP
S6.1		WaitCMDResponse	Substate when the PM is waiting for the FDC-Command.rsp
S6.2		SendingCMDResponse	Substate when the PM is waiting for completion of DL_SDN.req
S7	WaitComCycle		State when the PM is detecting the start edge of the communication cycle with a change of CMD-PDUs The destination state of the transition splits in two states depending on the transfer mode.
S7.0		Idle	Substate when the PM is waiting for the start signal for the next primary cycle
S7.1		WaitDL_GET_DATAcnf	Substate when the PM is waiting for the associated DL_GET_DATA.cnf to check its contents after requesting the DL_GET_DATA.req to the related DL-SAPID When the CMD Code and the WDT in the got PDU with the identical values to the previous ones, ARPM-FDCS transit to S7.0 and then waits for the next cycle without executing any process. On the other hand, when those values have changed, the PM has a transition to exit from this state.

9.2.3.2 Triggering events

Table 32 lists each trigger events of the ARPM-FDCS.

Table 32 – Trigger event descriptions of ARPM-FDCS

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		
E2	AR-Open.req	FSM ASE	DLSAPID, FDCSAPID	
E3	AR-Enable.req	FSM ASE	TransmissionMode	
E4	EVM-SyncEvent.ind	EVM ASE	NetClock	
E5	FDC-DataExchange.cnf	FDC ASE (Slave)	CMD-SDU	
E6	AR-ResetCycle.req	FDC ASE (Slave)		
E7	AR-StartComCycle.req	FDC ASE (Slave)	ComTime, DTMode, CMDForm	
E8	AR-SendCommand.rsp	FDC ASE (Slave)	ServiceStatus, RSP_ASDU	
E9	DL-READ-DATA.cnf	DLL	Result, , DLSDU	
E10	DL-WRITE-DATA.cnf	DLL	Result	
E11	DL_SDN.ind	DLL	SAP_ID, Node_ID,Length, DLSDU	
E12	DL_SDN.cnf	DLL	Result	

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E13	Error detected	FDC ASE (Slave)		See 8.2.7
E14	ComCycleStart	This object		Internal event from S7

9.2.3.3 Action descriptions at state transitions

Table 33 describes state transitions of the main SM of ARPM-FDCS. Moreover, Table 34 describes state transitions of the submachine of ARPM-FDCS.

Table 33 – Transitions of main SM of ARPM-FDCS

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1:AR-Reset.req / /*-- Clearing all attributes of the FDC-Slave AR --*/;	S0:Disabled
T2	S0:Disabled	E2:AR-Open.req / /*-- Initializing the FDC-Slave AR --*/; [a]	S0:Disabled
T3	S0:Disabled	E3:AR-Enable.req (comMode) [comMode == “cyclic”] / /*-- no-operation --*/;	S1:PrimaryCycle
T4	S0:Disabled	E3:AR-Enable.req (comMode) [comMode == “eventDriven”] / /*-- no-operation --*/;	S6:EventDriven
T5	S1:PrimaryCycle	E7:AR-StartComCycle / /*-- no-operation --*/;	S7:WaitComCycle
T6	S1:PrimaryCycle	The other events / /*-- state transition in the submachine --*/;	S1:PrimaryCycle
T7	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 1] / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Slave objects;	S2:FirstCycle
T8	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime == 2] / CycleScaleCounter++;	S5:LastCycle
T9	S2:FirstCycle	E4:EVM-SyncEvent.ind [ComTime > 2] / CycleScaleCounter++;	S4:MiddleCycle
T10	S2, S4, or S5	E6:AR-ResetCycle.req / /*-- no-operation --*/;	S1:PrimaryCycle
T11	S2:FirstCycle	The other events / /*-- state transition in the submachine --*/;	S2:FirstCycle
T12	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter < (ComTime-2)] / CycleScaleCounter++;	S4:MiddleCycle
T13	S4:MiddleCycle	E4:EVM-SyncEvent.ind [CycleScaleCounter == (ComTime-2)] / CycleScaleCounter++;	S5:LastCycle
T14	S4:MiddleCycle	The other events / /*-- state transition in the submachine --*/;	S4:MiddleCycle
T15	S5:LastCycle	E4:EVM-SyncEvent.ind / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Slave objects;	S2:FirstCycle
T16	S5:LastCycle	The other events / /*-- state transition in the submachine --*/;	S5:LastCycle
T17	S6:EventDriven	The other events / /*-- state transition in the submachine --*/;	S6:EventDriven
T18	S7:WaitComCycle	E14:ComCycleStart [DTMode == Single] / CycleScaleCounter = 0; AR-CycleEvent.ind to FDC Slave objects;	S2:FirstCycle
T19	S7:WaitComCycle	E14:ComCycleStart [DTMode == Dual] / CycleScaleCounter = ComTime-1;	S5:LastCycle
T20	S7:WaitComCycle	The other events / /*-- state transition in the submachine --*/;	S7:WaitComCycle

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

Table 34 – Transitions of submachine of ARPM-FDCS

T#	Source State	Event (arguments) [conditions] / action	Target State
T6.0 (T3)	(S0)	/entry / /*-- no-operation --*/;	S1.0:Idle
T6.1	S1.0:Idle	E4:EVM-SyncEvent.ind / DL_GET_DATA.req ;	S1.1:WaitDL_GET_DATAcnf
T6.2	S1.1:WaitDL_GET_DATAcnf	E9:DL_GET_DATA.cnf (csdu) / FDC-DataExchange.req (csdu);	S1.2:WaitFDC-DataExchangecnf
T6.3	S1.2:WaitFDC-DataExchangecnf	E5:FDC-DataExchange.cnf (rsdu) / DL_SET_DATA.req (rsdu);	S1.3:WaitDL_SET_DATAcnf
T6.4	S1.3:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S1.0:Idle
T11.0 (T7, T15, T18)	(S2, S5,)	/entry / DL_GET_DATA.req ;	S2.1:WaitDL_GET_DATAcnf
T11.1	(S7)	/entry / FDC-DataExchange.req (lastCsdu);	S2.2: WaitFDC-DataExchangecnf
T11.2	S2.1:WaitDL_GET_DATAcnf	E9: DL_GET_DATA.cnf (csdu) / if (DTMode == “Dual”) { /*-- Error detection and PDU comparison process -- with lastCsdu and csdu --*/; } else {/*-- no-operation --*/;} FDC-DataExchange.req (csdu); lastCsdu = csdu;	S2.2: WaitFDC-DataExchangecnf
T11.3	S2.2: WaitFDC-DataExchangecnf	E5:FDC-DataExchange.cnf (rsdu) / DL_SET_DATA.req (rsdu);	S2.3: :WaitDL_SET_DATAcnf
T11.4	S2.3:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S.2.0:Idle
T14.0 (T10)	(S2:FirstCycle)	/entry / FDC-DataExchange.req (csdu1);	S4.1: WaitFDC-DataExchangecnf
T14.1	S4.1: WaitFDC-DataExchangecnf	E5:FDC-DataExchange.cnf (rsdu) / DL_SET_DATA.req (rsdu);	S4.2: :WaitDL_SET_DATAcnf
T14.2	S4.2:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S.4.0:Idle
T16.0	(S2, S4)	/entry / DL_GET_DATA.req;	S5.1:WaitDL_GET_DATAcnf
T16.1	(S7)	/entry / /*-- no-operation --*/;	S5.0:Idle
T16.2	S5.1:WaitDL_GET_DATAcnf	E9: DL_GET_DATA.cnf (csdu) / FDC-DataExchange.req (lastCsdu); lastCsdu = csdu;	S5.2:WaitFDC-DataExchangecnf
T16.3	S5.2:WaitFDC-DataExchangecnf	E5: FDC-DataExchange.cnf (rsdu) / DL_SET_DATA.req (rsdu);	S5.3:WaitDL_SET_DATAcnf
T16.4	S5.3:WaitDL_SET_DATAcnf	E10:DL_SET_DATA.cnf (+) / /*-- no-operation --*/;	S5.0:Idle
T17.0 (T4)	(S0)	/entry: / /*-- no-operation --*/;	S6.0:WaitRequestCMD
T17.1	S6.0:WaitCMDIndication	E11: DL_SDN.ind (SAP_ID, Node_ID, Length, DLSDU) / AR-SendCommand.ind (csdu);	S6.1:WaitCMDResponse
T17.2	S6.1:WaitCMDResponse	E8: AR-SendCmmand.rsp (serviceStatus, rsdu) / DL_SDN.req (SAP_ID, Node_ID,Length, DLSDU);	S6.2:SendingCMD Response
T17.3	S6.2:SendingCMD Response	E12: DL_SDN.cnf (+) / /*-- no-operation --*/;	S6.0:WaitCMDIndication
T17.4	S6.2:SendingCMD Response	E13:Communication time-out / /*-- Communication alarm processing/resume processing --*/;	S6.0:WaitCMDIndication
T20.0		/entry / /*-- no-operation --*/;	S7.0:Idle
T20.1	S7.0:Idle	E4: EVM-SyncEvent.ind	S7.1:

T#	Source State	Event (arguments) [conditions] / action	Target State
		/ DL_GET_DTAT.req;	WaitDL_GET_DAT Acnf
T20.2	S7.1: WaitDL_GET_DAT Acnf	E9:DL_GET_DATA.cnf (csdu) [csdu == lastCsdu] / lastCsdu = csdu;	S7.0:Idle
T20.3	S7.1: WaitDL_GET_DAT Acnf	E9:DL_GET_DATA.cnf (csdu) [csdu != lastCsdu] / /*-- signal E14:StartComCycle to the main machine --*/;	exit from the submachine
NOTE "lastCsdu" is a local variable.			

9.2.4 ARPM for FDC Monitor (ARPM-FDCMN)

9.2.4.1 State descriptions

Figure 29 shows the ARPM-FDCMN statechart diagram, and Table 35 describes each state of the ARPM-FDCMN.

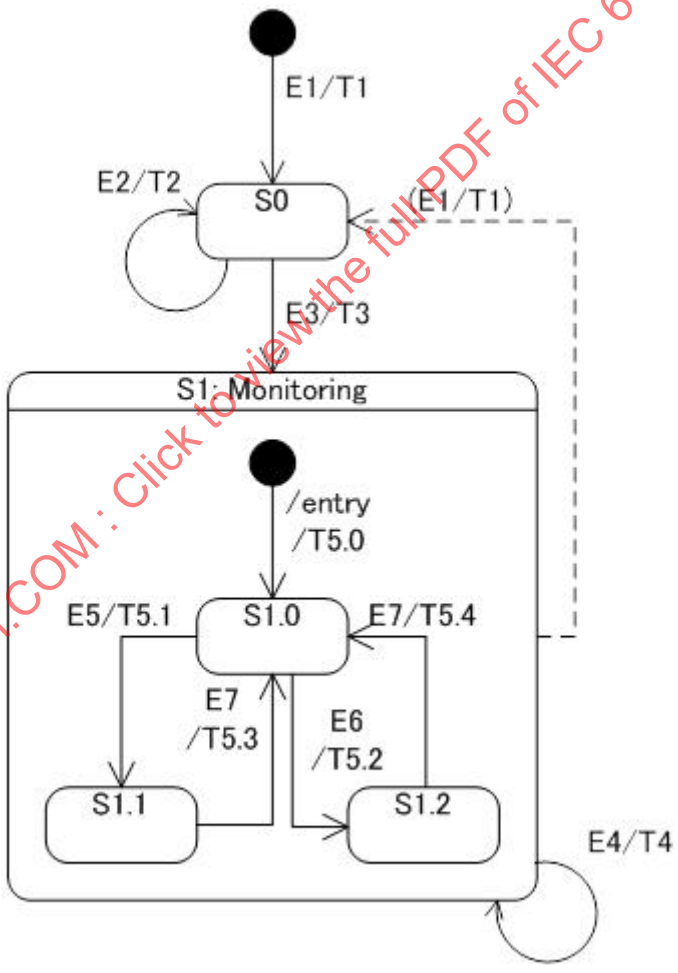


Figure 29 – Statechart diagram of ARPM-FDCMN

Table 35 – State descriptions of ARPM-FDCMN

S#	State	Substate	Description
S0	Disabled	-	State when the FDC Monitor AR Class has just 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 FDC Monitor AR object
S1	Monitoring	-	State when the PM is activated to monitor the CMD-PDU or the RSP-PDU
S1.0		Idle	Substate when the PM is waiting for an FDC ASE's request to get CMD-PDU or RSP-PDU by the AR-GetCMD.req or the AR-GetRSP.req
S1.1		WaitCMDData	Substate when the PM is waiting to receive CMD-PDU from the other station through the DLL
S1.2		WaitRSPData	Substate when the PM is waiting to receive RSP-PDU from the other station through the DLL

9.2.4.2 Triggering events

Table 36 lists each trigger events of the ARPM-FDCMN.

Table 36 – Trigger event descriptions of ARPM-FDCMN

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		
E2	AR-Open.req	FSM ASE	DLSAPID, ASESAPID	
E3	AR-Enable.req	FSM ASE	TransmissionMode	
E4	EVM-SyncEvent.ind	EVM ASE	NetworkClock	
E5	AR-GetCMD.req	FDC ASE (Monitor)	APID	
E6	AR-GetRSP.req	FDC ASE (Monitor)	APID	
E7	DL-READ-DATA.cnf	DLL	Result, DLSDU	

9.2.4.3 Action descriptions at state transitions

Table 37 describes state transitions of the main SM of ARPM-FDCMN. Moreover, Table 38 describes state transitions of the submachine of ARPM-FDCMN.

Table 37 – Transitions of main SM of ARPM-FDCMN

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1:AR-Reset.req / /*-- Clearing all attributes of the FDC-Monitor AR --*/;	S0:Disabled
T2	S0:Disabled	E2:AR-Open.req / /*-- Initializing the FDC-Monitor AR --*/; [a]	S0:Disabled
T3	S0:Disabled	E3:AR-Enable.req / /*-- no-operation --*/;	S1:Monitoring
T4	S1:Monitoring	E4: EVM-SyncEvent.ind	S1:Monitoring

T#	Source State	Event (arguments) [conditions] / action	Target State
		/ /*-- no-operation --*/;	
T5	S1:Monitoring	The other events / /*-- state transition in the submachine --*/;	S1:Monitoring
[a] The detailed process depends on the system implementation.			

Table 38 – Transitions of submachine of ARPM-FDCMN

T#	Source State	Event (arguments) [conditions] / action	Target State
T5.0 (T3)	S0	/entry / /*-- no-operation --*/;	S1.0:Idle
T5.1	S1.0:Idle	E5:AR-GetCMD.req(DL-NodeID) / /*-- Generate SAP_ID from DL-NodeID --*/; DL_GET_DATA.req(SAP_ID);	S1.1:WaitCMDData
T5.2	S1.0:Idle	E6:AR-GetRSP.req(DL-NodeID) / /*-- Generate SAP_ID from DL-NodeID --*/; DL_GET_DATA.req(SAP_ID);	S1.2:WaitRSPData
T5.3	S1.1:WaitCMDData	E7:DL_GET_DATA.cnf(Node_ID, DLSDU, Result) / /*-- Generate CMD_ASDU from DLSDU --*/; AR-GetCMD.rsp (ServiceStatus, CMD_ASDU);	S1.0:Idle
T5.4	S1.2:WaitRSPData	E7:DL_GET_DATA.cnf(Node_ID, DLSDU, Result) / /*-- Generate RSP_ASDU from DLSDU. --*/; AR-GetRSP.rsp (ServiceStatus, RSP_ASDU);	S1.0:Idle

9.3 ARPM for MSG ASE (ARPM-MSG)

9.3.1 State descriptions

The MSGPM manages the transaction of messages. On the other hand, this ARPM only manages each activity of transmission or reception.

Figure 30 shows the ARPM-MSG statechart diagram, and Table 39 describes each state of the ARPM-MSG.

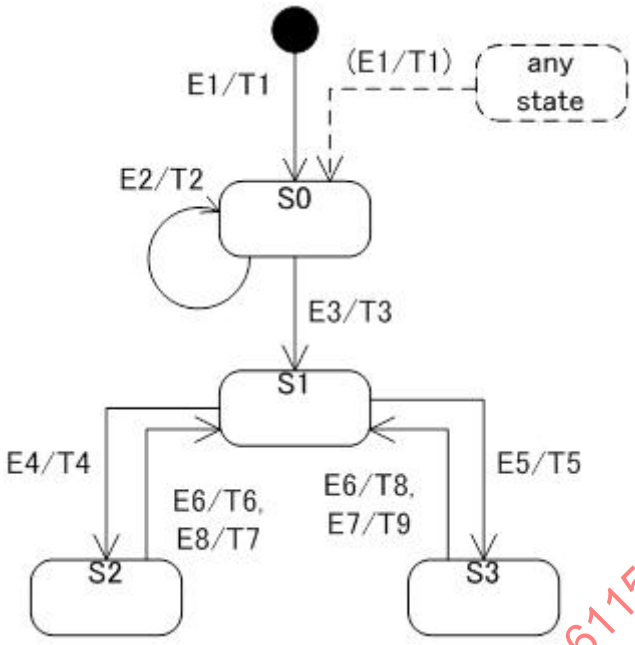


Figure 30 – Statechart diagram of ARPM-MSG

Table 39 – State descriptions of ARPM-MSG

S#	State	Substate	Description
S0	Disabled		State when the Message AR Class has just 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 Message AR object
S1	Ready		State when the PM is waiting for the service request, the AR-SendMessage.req or the AR-ReceiveMessage.req, from the MSG ASE
S2	SendingMSG		State when the PM is waiting for the completion of the transmission, DL-SDA.cnf, after receiving AR-SendMessage.req
S3	ReceivingMSG		State when the PM is waiting for message reception, DL-SDA.ind, after receiving AR-ReceiveMessage.req

9.3.2 Triggering events

Table 40 lists each trigger events of the ARPM-MSG.

Table 40 – Trigger event descriptions of ARPM-MSG

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E1	AR-Reset.req	FSM ASE		
E2	AR-Open.req	FSM ASE	DLSAPID, MSGSAPID	
E3	AR-Enable.req	FSM ASE		
E4	AR-SendMessage.req	MSG ASE	DPID, Length, MSGService-PDU	
E5	AR-ReceiveMessage.req	MSG ASE	SPID, MaxLength	
E6	AR-AbortMessage.req	MSG ASE		
E7	DL-SDA.ind	DLL	SAP_ID, NodeID, Length, DLSDU	

E#	Trigger event: Primitive or condition	Event source	Associated parameters	Description
E8	DL-SDA.cnf	DLL	Result	

9.3.3 Action descriptions at state transitions

Table 41 describes state transitions of the state machine of ARPM-MSG.

Table 41 – Transitions of ARPM-MSG

T#	Source State	Event (arguments) [conditions] / action	Target State
T1	(any state)	E1: AR-Reset.req / /*-- Clearing all attributes of the Message-AR --*/;	S0:Disabled
T2	S0:Disabled	E2: AR-Open.req (AREPTYPE, DLSAPID) / /*-- Initializing the Message-AR --*/; [a]	S0:Disabled
T3	S0:Disabled	E3: AR-Enable.req / /*-- no-operation --*/;	S1:Ready
T4	S1:Ready	E4: AR-SendMessage.req (TID, Length, MSGReq-PDU) / /*-- Generate DLSDU from MSGRsp-PDU. --*/; DL_SDA.req (DLSDU);	S2:SendingMSG
T5	S1:Ready	E5: AR-ReceiveMessage.req / /*-- no-operation --*/;	S3:ReceivingMSG
T6	S2:SendingMSG	E6: AR-AbortMessage.req (SAPID, TID) / /*-- no-operation --*/;	S1:Ready
T7	S2:SendingMSG	E8: DL_SDA.cnf (+) (Result) / /*-- no-operation --*/;	S1:Ready
T8	S3:ReceivingMSG	E6: AR-AbortMessage.req (SAPID, TID) / /*-- no-operation --*/;	S1:Ready
T9	S3:ReceivingMSG	E7: DL_SDA.ind (SAPID, NodeID, Length, DLSDU) / /*-- Generate MSGRsp-PDU from DLSDU. --*/; AR-ReceiveMessage.cnf (+) (TID, Length, MSGRsp-PDU);	S1:Ready

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

10 DLL mapping protocol machine (DMPM)

The DMPM handles the DLL interface of the AR ASE. However, it does not exist as an independent protocol machine in the Type 24 FAL, because its function is executed directly by the ARPM.

Annex A
(informative)

Device profile and FDC command sets

Table A.1 shows an example of the device profiles. The device profile provides FAL users a command set of the _FDCServicePDU for a certain application field.

Table A.1 – Example of registered device profiles

Range	Category	Profile Code	Profile name
‘00’H to ‘0F’H	System common profiles	‘00’H	Legacy command set
		’01’H	Minimum command set for device configuration
		…	Reserved
‘10’H to ‘1F’H	Servo drive profiles	‘10’H	Standard type servo drive
		‘11’H	High resolution control type servo drive
		…	Reserved
‘20’H to ‘2F’H	Inverter drive profiles	‘20’H	Standard type inverter drive
		…	Reserved
‘30’H to ‘3F’H	I/O device profiles	‘30’H	Standard type I/O device
		…	Reserved
‘40’H to ‘7F’H	Reserved	n/a	n/a
‘80’H to ‘FF’H	Reserved	n/a	n/a

Table A.2 shows an example of the command set. The contents are a set of command codes of the profile ‘00’H, or the legacy but most common command set.

Table A.2 – Example command list of the profile ‘00’H

Code	+0	+1	+2	+3	Description
’00’H	NOP	PRM_RD	PRM_WR	ID_RD	Device common commands
‘04’H	CONFIG	ALM_RD	ALM_CLR	n/a	
‘08’H	n/a	n/a	n/a	n/a	
‘0C’H	n/a	SYNC_SET	CONNECT	DISCONNECT	
‘10’H	n/a	n/a	n/a	n/a	
‘14’H	n/a	n/a	n/a	n/a	
‘18’H	n/a	n/a	n/a	PPRM_RD	
‘1C’H	PPRM_WR	n/a	n/a	n/a	
’20’H	POS_SET	BRK_ON	BRK_OFF	SENS_ON	Servo commands, inverter commands, and so on
‘24’H	SENS_OFF	HOLD	n/a	n/a	
‘28’H	n/a	n/a	n/a	n/a	
‘2C’H	n/a	n/a	n/a	n/a	
’30’H	SMON	SV_ON	SV_OFF	n/a	
‘34’H	INTERPOLATE	POSING	FEED	n/a	
‘38’H	LATCH	EX_POSING	ZRET	n/a	
‘3C’H	VELCTRL	TRQCTRL	ADJ	SVCTRL	
‘40’H	INV_CTRL	INV_IO	n/a	n/a	
…	…	…	…	…	Omitted
‘50’H	DATA_RWR	DATA_RWS	n/a	n/a	
…	…	…	…	…	Omitted

Annex B
(normative)

Virtual memory space and Device Information

B.1 Overview

The Device Information shall be defined and embedded as data blocks on virtual memory space (see Figure B.1). Therefore, every communication system users can access the information through the network about any vendor's products in a standardized manner.

The Device Information may be also provided as a script file with a specific manner. Such a file is called the "Mechatronic Device Information" (MDI) file in the Type 24 FAL. But the script specification is out of the scope for this standard.

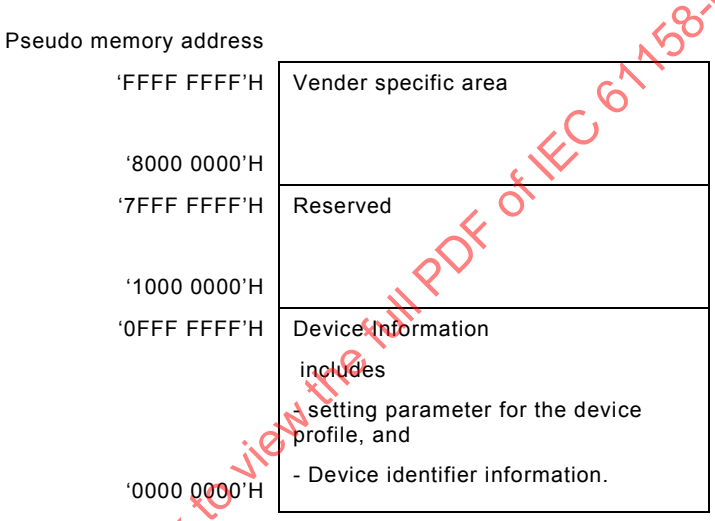


Figure B.1 – Memory map of virtual memory space

B.2 Device Information

B.2.1 Device identifier area structure

Device identifier information (Device ID) area is a part of the Device Information in the virtual memory space. Therefore it shall be able to be read out by using either ID_RD-CMD-PDU or MEM_RD-CMD-PDU.

Figure B.2 shows the memory map of the device ID area.

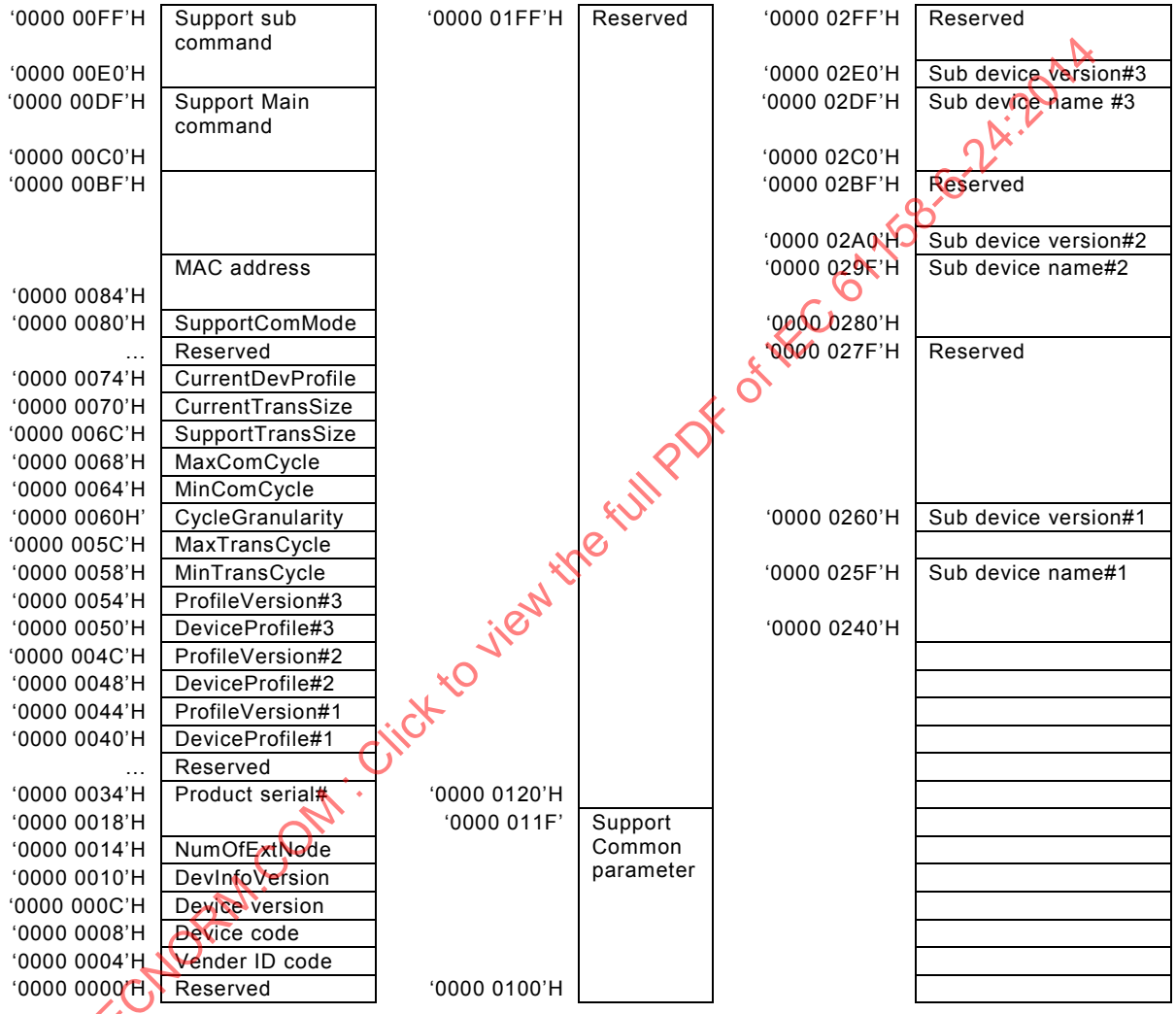


Figure B.2 – Memory map of device ID area

B.2.2 Detail specifications of device IDs

Table B.1 describes detail specifications of each device IDs.

Table B.1 – Specifications of device IDs

<table>
<tr><th>ID CODE</th><th>Contents</th><th>Data size</th><th>Data type</th><th>provision</th></tr>
<tr><td>‘0001’H</td><td>Vendor ID code</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: ID code that identifies the vendor</td></tr>
<tr><td>‘0002’H</td><td>Device code</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: code number that represents each device model
This number shall be unique to each product series within a vender ID code.
The number ‘FFFF FFFF’H shall not be specified because of the reserved code for the system use.</td></tr>
<tr><td>‘0003’H</td><td>Device version</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: version information of the device</td></tr>
<tr><td>‘0004’H</td><td>Version of Device Information file</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: version of the Mechatronic Device Information (MDI) file provided by the device vendor
<table>
<tr><td>Bit number</td><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr>
<tr><td>Version information</td><td colspan="8">Additional code</td></tr>
</table>
<table>
<tr><td>Bit number</td><td>bit15</td><td>bit14</td><td>bit13</td><td>bit12</td><td>bit11</td><td>bit10</td><td>bit9</td><td>bit8</td></tr>
<tr><td>Version information</td><td colspan="4">Major version</td><td colspan="4">Minor version</td></tr>
</table>
Major version: Sequential number updated when substantial changes have been made to the MDI file due to improvement and/or modification of function;
Minor version: Sequential number updated when changes have been made to the MDI file due to addition and/or modification of a function in a small scale;
Additional code: not a kind of version but a vendor specific code number for a history of the MDI file;
bit16 : Reserved(0).</td></tr>
<tr><td>‘0005’H</td><td>Number of Extended addresses</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: number of the extended address to be needed or a number of logical slave APs in a physical slave device
The extended address can be used to allow a station address to have more than one slave APs.</td></tr>
<tr><td>‘0006’H</td><td>Serial number of product</td><td>32 octets</td><td>IA5String (Delimiter ’00’H)</td><td>Optional</td></tr>
<tr><td></td><td colspan="4">Description:serial number that specify the individual product</td></tr>
<tr><td>‘0010’H</td><td>Supported Device Profile 1 (Primary)</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: code of the primary device profile implemented in the device
For a device that can accept more than one profile, additional codes can be retrieved from the secondary one (‘0012’H) and the third one (‘0014’H). In that case, the priority for the profiles will be as shown below:
Device Profile 1 (primary) (this ID_CODE);
Device Profile 2 (secondary) (‘0012’H);
Device Profile 3 (tertiary) (‘0014’H).</td></tr>
<tr><td>‘0011’H</td><td>Supported Profile version 1 (primary)</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
</table>

<table>
<tr><th>ID CODE</th><th>Contents</th><th>Data size</th><th>Data type</th><th>provision</th></tr>
<tr><td></td><td colspan="4">Description: version of the primary device profile

For a device that can accept more than one profile, additional versions can be retrieved from the secondary one (‘0013’H) and the third one (‘0015’H).

<table>
<tr><td>Bit number</td><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr>
<tr><td>Version information</td><td colspan="8">Minor version</td></tr>
</table>

<table>
<tr><td>Bit number</td><td>bit15</td><td>bit14</td><td>bit13</td><td>bit12</td><td>bit11</td><td>bit10</td><td>bit9</td><td>bit8</td></tr>
<tr><td>Version information</td><td colspan="8">Major version</td></tr>
</table>

Major version: sequential number updated when substantial changes have been made to the profile specification

Minor version: sequential number updated when changes in a small scale have been made to the profile specification</td></tr>
<tr><td rowspan="2">‘0012’H</td><td>Supported Device profile 2 (secondary)</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: code of the secondary device profile implemented in the device

If the device can accept only the primary one, the code shall be set to ‘00FF’H (unsupported code).</td></tr>
<tr><td rowspan="2">‘0013’H</td><td>Supported Profile version 2 (secondary)</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: version of the secondary device profile

When the device does not handle the device profile 2, the code shall be set to ‘0000’H.</td></tr>
<tr><td rowspan="2">‘0014’H</td><td>Supported Device profile 3 (tertiary)</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: code of the tertiary device profile implemented in the device

If the device cannot accept the tertiary one, the code shall be set to ‘00FF’H (unsupported code).</td></tr>
<tr><td rowspan="2">‘0015’H</td><td>Supported Profile version 3 (tertiary)</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: version of the tertiary device profile

When the device does not handle the device profile 3, the code shall be set to ‘0000’H.</td></tr>
<tr><td rowspan="2">‘0016’H</td><td>Minimum transmission cycle</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: the minimum value of the selectable transmission cycle by the unit 0,01 µs for the device under the condition of the granularity level (‘0018H’)

Example

Setting value: 3 125 (decimal) means the minimum transmission cycle: 31,25 ms.</td></tr>
<tr><td rowspan="2">‘0017’H</td><td>Maximum transmission cycle</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: the maximum value of the selectable transmission cycle by the unit 0,01 µs for the device under the condition of the granularity level (‘0018H’)

Example

Setting value: 800 000 (decimal) means the maximum transmission cycle: 8 ms</td></tr>
<tr><td rowspan="2">‘0018’H</td><td>Granularity level of transmission cycle</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td colspan="4">Description: the combination of the step width of the selectable transmission cycle for the device

There are four levels as shown in the followings.

‘0000’H: 31,25/ 62,5/ 125/ 250/ 500 [µs], and 2 ms to 64 ms by 2 ms

‘0001’H: 31,25/ 62,5/ 125/ 250/ 500 [µs], and 1 ms to 64 ms by 1 ms

‘0002’H: 31,25/ 62,5/ 125/ 250/ 500 [µs], and 1 ms to 64 ms by 0,5 ms

‘0003’H: 31,25/ 62,5/ 125/ 250/ 500/ 750 [µs], and 1 ms to 64 ms by 0,5 ms</td></tr>
<tr><td>‘0019’H</td><td>Minimum communication cycle</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
</table>

<table>
<tr><th>ID CODE</th><th>Contents</th><th>Data size</th><th>Data type</th><th>provision</th></tr>
<tr><td></td><td colspan="4">Description: the minimum value of the selectable communication cycle by the unit 0,01 µs for the device under the condition of the granularity level (‘0018H’)
Example
Setting value: 3 125 (decimal) means the minimum communication cycle: 31,25 ms.</td></tr>
<tr><td>‘001A’H</td><td>Maximum communication cycle</td><td>4 octets</td><td>Unsigend32</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: the maximum value of the selectable communication cycle by the unit 0,01 µs for the device under the condition of the granularity level (‘0018H’)
Example
Setting value: 800 000 (decimal) means the maximum communication cycle: 8 ms.</td></tr>
<tr><td>‘001B’H</td><td>Number of transmittable octets</td><td>4 octets</td><td>BitArray</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: supported octet size of the _FDCServicePDU for the device
Each selectable size option is mapped to a bit as follows (supported: 1, not-supported: 0).
<table>
<tr><th>bit7</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th></tr>
<tr><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>64 octets</td><td>48 octets</td><td>32 octets</td><td>16 octets</td><td>8 octets</td></tr>
</table>
Bit8～bit31: reserved (0).</td></tr>
<tr><td>‘001C’H</td><td>Number of transmitted octets (current setting)</td><td>4 octets</td><td>BitArray</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: selected octet size of the _FDCServicePDU under the condition of the number of transmittable octets (‘001B’H)
Each selectable size option is mapped to a bit as follows (selected: 1, not-selected: 0).
<table>
<tr><th>bit7</th><th>bit6</th><th>bit5</th><th>bit4</th><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th></tr>
<tr><td>Reserved</td><td>Reserved</td><td>Reserved</td><td>64 octets</td><td>48 octets</td><td>32 octets</td><td>16 octets</td><td>8 octets</td></tr>
</table>
bit8～bit31: reserved (0).</td></tr>
<tr><td>‘0020’H</td><td>Supported Communication mode List</td><td>4 octets</td><td>BitArray</td><td>Required</td></tr>
<tr><td></td><td colspan="4">Description: the list of the supported transmission mode and messages communication mode for the device
Each communication mode is assigned to a bit as follows (supported: 1, not-supported: 0).
<table>
<tr><th>bit3</th><th>bit2</th><th>bit1</th><th>bit0</th></tr>
<tr><td>Ethernet communication</td><td>Message communication</td><td>Cyclic communication</td><td>Event-driven communication</td></tr>
</table>
bit4 ~bit31: reserved(0)</td></tr>
<tr><td>‘0021’H</td><td>MAC address</td><td>8 octets</td><td>Unsigend64</td><td>Optional</td></tr>
<tr><td></td><td colspan="4">Description: ethernet MAC address for the device
The valid value should be only set to the products that support Ethernet communication.</td></tr>
<tr><td>‘0030’H</td><td>Supported main command list</td><td>32 octets</td><td>BitArray</td><td>Required</td></tr>
</table>

<table>
<tr><th>ID CODE</th><th>Contents</th><th>Data size</th><th>Data type</th><th>provision</th></tr>
<tr><td></td><td>Description: the list of supported main commands for the device
Each command code is assigned to a corresponding bit position as shown in the following tables.
Command codes of bit32 to bit255 are dependent on the selected device profile.
Each bit shows
0 : not-supported command, and
1: supported command.
<table>
<tr><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr>
<tr><td>Reserved (0)</td><td>ALM_CLR</td><td>ALM_RD</td><td>CONFIG</td><td>ID_RD</td><td>PRM_WR</td><td>PRM_RD</td><td>NOP</td></tr>
</table>
<table>
<tr><td>bit15</td><td>bit14</td><td>bit13</td><td>bit12</td><td>bit11</td><td>bit10</td><td>bit9</td><td>bit8</td></tr>
<tr><td>DISCONNECT</td><td>CONNECT</td><td>SYNC_SET</td><td>Reserved (0)</td><td>Reserved (0)</td><td>Reserved (0)</td><td>Reserved (0)</td><td>Reserved (0)</td></tr>
</table>
Bit16 to bit23: reserved (0).
<table>
<tr><td>Bit31</td><td>bit30</td><td>bit29</td><td>bit28</td><td>bit27</td><td>bit26</td><td>bit25</td><td>bit24</td></tr>
<tr><td>Reserved (0)</td><td>MEM_WR</td><td>MEM_RD</td><td>PPRM_WR</td><td>PPRM_RD</td><td>Reserved (0)</td><td>Reserved (0)</td><td>Reserved (0)</td></tr>
</table>
Bit32 to bit255: dependent on the device profile.</td></tr>
<tr><td>‘0038’H</td><td>Supported sub command list</td><td>32 octets</td><td>BitArray</td><td>Required</td></tr>
<tr><td></td><td>Description: the list of supported subcommands for the device
Each command code is assigned to a corresponding bit position as shown in the following tables.
Command codes of bit32 to bit255 are dependent on the selected device profile.
Each bit shows
0 : not-supported command, and
1: supported command.
<table>
<tr><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr>
<tr><td>Reserved (0)</td><td>Reserved (0)</td><td>ALM_RD</td><td>Reserved (0)</td><td>Reserved (0)</td><td>PRM_WR</td><td>PRM_RD</td><td>NOP</td></tr>
</table>
Bit8 – bit26 reserved (0).
<table>
<tr><td>bit31</td><td>bit30</td><td>bit29</td><td>bit28</td><td>bit27</td><td>bit26</td><td>bit25</td><td>bit24</td></tr>
<tr><td>Reserved (0)</td><td>Reserved (0)</td><td>Reserved (0)</td><td>PPRM_WR</td><td>PPRM_RD</td><td>Reserved (0)</td><td>Reserved (0)</td><td>Reserved (0)</td></tr>
</table>
Bit32 – bit255: dependent on the device profile.</td></tr>
<tr><td>‘0040’H</td><td>Supported Common parameter list</td><td>32 octets</td><td>BitArray</td><td>Required</td></tr>
</table>

<table>
<tr><th>ID CODE</th><th>Contents</th><th>Data size</th><th>Data type</th><th>provision</th></tr>
<tr><td></td><td colspan="4">Description: the list of supported common parameters for the device
Each parameter code is assigned to a corresponding bit position as shown in the following tables.
The means of common parameter codes is dependent on the device profile.
Each bit shows
0 : not-supported parameter code, and
1: supported parameter code.
<table>
<tr><td>bit7</td><td>bit6</td><td>bit5</td><td>bit4</td><td>bit3</td><td>bit2</td><td>bit1</td><td>bit0</td></tr>
<tr><td>‘07’H</td><td>‘06’H</td><td>‘05’H</td><td>‘04’H</td><td>‘03’H</td><td>‘02’H</td><td>‘01’H</td><td>‘00’H</td></tr>
</table>
<table>
<tr><td>bit15</td><td>bit14</td><td>bit13</td><td>bit12</td><td>bit11</td><td>bit10</td><td>bit9</td><td>bit8</td></tr>
<tr><td>‘0F’H</td><td>‘0E’H</td><td>‘0D’H</td><td>‘0C’H</td><td>‘0B’H</td><td>‘0A’H</td><td>‘09’H</td><td>‘08’H</td></tr>
</table>
(Bit16 to bit239 are omitted.)
<table>
<tr><td>bit247</td><td>bit246</td><td>bit245</td><td>bit244</td><td>bit243</td><td>bit242</td><td>bit241</td><td>bit240</td></tr>
<tr><td>‘F7’H</td><td>‘F6’H</td><td>‘F5’H</td><td>‘F4’H</td><td>‘F3’H</td><td>‘F2’H</td><td>‘F1’H</td><td>‘F0’H</td></tr>
</table>
<table>
<tr><td>bit255</td><td>bit254</td><td>bit253</td><td>bit252</td><td>bit251</td><td>bit250</td><td>bit249</td><td>bit248</td></tr>
<tr><td>‘FF’H</td><td>‘FE’H</td><td>‘FD’H</td><td>‘FC’H</td><td>‘FB’H</td><td>‘FA’H</td><td>‘F9’H</td><td>‘F8’H</td></tr>
</table></td></tr>
<tr><td rowspan="2">‘0080’H</td><td>Name of Main device</td><td>32 octets</td><td>IA5String (Delimiter "00")</td><td>Optional</td></tr>
<tr><td colspan="4">Description: name of the device with readable characters
It means the delegate name of the main device when a physical device contains plural logical APs or subdevices.
NOTE A user should not use this data but the device code (ID CODE: ‘0002’H) to identify the device type or model.</td></tr>
<tr><td rowspan="2">0090H</td><td>Name of subdevice 1</td><td>32 octets</td><td>IA5String (Delimiter ’00’H)</td><td>Optional</td></tr>
<tr><td colspan="4">Description: name of the first subdevice with readable characters
The subdevice is specified in the product specification.</td></tr>
<tr><td rowspan="2">0098H</td><td>Version of subdevice 1</td><td>4 octets</td><td>Unsigend32</td><td>Optional</td></tr>
<tr><td colspan="4">Description: version information of the first subdevice in the device</td></tr>
<tr><td rowspan="2">00A0H</td><td>Name of sub device 2</td><td>32 octets</td><td>IA5String (Delimiter ’00’H)</td><td>Optional</td></tr>
<tr><td colspan="4">Description: name of the second subdevice with readable characters
The subdevice is specified in the product specification.</td></tr>
<tr><td rowspan="2">00A8H</td><td>Version of subdevice 2</td><td>4 octets</td><td>Unsigend32</td><td>Optional</td></tr>
<tr><td colspan="4">Description: version information of the second subdevice in the device</td></tr>
<tr><td rowspan="2">00B0H</td><td>Name of sub device 3</td><td>32 octets</td><td>IA5String (Delimter ’00’H)</td><td>Optional</td></tr>
<tr><td colspan="4">Description: name of the third subdevice with readable characters
The subdevice is specified in the product specification.</td></tr>
<tr><td rowspan="2">00B8H</td><td>Version of subdevice 3</td><td>4 octets</td><td>Unsigend32</td><td>Optional</td></tr>
<tr><td colspan="4">Description: version information of the third subdevice in the device</td></tr>
<tr><td>00BCH
~
00BFH</td><td colspan="4">Reserved</td></tr>
<tr><td>00C0H
~
00FFH</td><td colspan="4">Vendor-specific area</td></tr>
</table>

Annex C
(informative)

Basic message function

The framework of the message PDUs are described in 4.2.3 (also see below). But, more details of message PDU syntax should be defined by the FAL user in case of the Type 24 FAL.

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

In this annex, an example command set for function code:0x42 is shown in Table C.1.

Table C.1 – Example of message command set

Function code	Command Code	Command description
0x42	0x01	To read data from a single memory block
	0x02	To write data to a single memory block
	0x03	To read data from multiple memory blocks
	0x04	To write data to multiple memory blocks
	0x06	To write data to memory with bit mask pattern

Bibliography

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

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

ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN-1): Specification of basic notation*

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

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

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-24: Spécification du protocole de la couche application – Éléments de type 24

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

L'attention est attirée sur le fait que l'utilisation du type de protocole associé est restreinte par les détenteurs des droits de propriété intellectuelle. En tout état de cause, l'engagement de renonciation partielle aux droits de propriété intellectuelle pris par les détenteurs de ces droits autorise l'utilisation d'un type de protocole de couche avec les autres protocoles de couche du même type, ou dans des combinaisons avec d'autres types autorisées explicitement par les détenteurs des droits de propriété intellectuelle pour ce type.

NOTE Les combinaisons de types de protocoles sont spécifiées dans la CEI 61784-1 et la CEI 61784-2.

La Norme internationale CEI 61158-6-24 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
65C/764/FDIS	65C/774/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61158, publiées sous le titre général *Réseaux de communication industriels – Spécifications des bus de terrain*, peut être consultée sur le site Web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

INTRODUCTION

La présente partie de la CEI 61158 fait partie d'une série élaborée pour faciliter l'interconnexion des composants de systèmes d'automatisation. Elle renvoie aux autres normes de l'ensemble défini par le modèle de référence de bus de terrain "à trois couches" décrit dans la CEI 61158-1.

Le protocole d'application fournit le service d'application au moyen des services disponibles au niveau de la couche Liaison de données ou de la couche immédiatement inférieure. Le principal objectif de la présente Norme est de définir un ensemble de règles de communication, exprimées en termes de procédures qu'ont à suivre les entités d'application (Application Entity, AE) homologues au moment de la communication. Ces règles de communication visent à fournir une base saine pour le développement, dans divers buts:

- servir de guide pour les ingénieurs d'application et les concepteurs;
- dans une optique d'utilisation lors de l'essai et de l'achat de matériel;
- dans le cadre d'un accord pour l'admission de systèmes dans l'environnement de systèmes ouverts;
- en tant que précision apportée à la compréhension des communications prioritaires dans le modèle OSI.

La présente Norme traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automatisation. Grâce à l'utilisation conjointe de la présente Norme avec d'autres normes entrant dans les modèles de référence OSI ou de bus de terrain, des systèmes autrement incompatibles peuvent fonctionner dans toute combinaison.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-24: Spécification du protocole de la couche application – Éléments de type 24

1 Domaine d'application

1.1 Généralités

La couche Application de bus de terrain (Fieldbus Application Layer, FAL) procure aux programmes de l'utilisateur un moyen d'accès à l'environnement de communication des bus de terrain. À cet égard, la FAL peut être considérée comme une "fenêtre entre programmes d'application correspondants".

La présente Norme donne les éléments communs visant à assurer les communications de messagerie de base prioritaires et non prioritaires entre les programmes d'application d'un environnement d'automatisation et un matériel spécifique au bus de terrain de Type 24. Le terme "prioritaires" est utilisé pour indiquer la présence d'une fenêtre temporelle, dans laquelle une ou plusieurs actions spécifiées sont à réaliser selon un certain niveau de certitude. La non-réalisation des actions spécifiées dans la fenêtre temporelle induit un risque de défaillance des applications qui demandent ces actions, avec les risques afférents pour l'équipement, les installations et éventuellement la vie humaine.

La présente Norme définit de manière abstraite les caractéristiques visibles en externe offertes par la couche d'application de bus de terrain de Type 24 en termes

a) de syntaxe abstraite définissant les unités de données du protocole de la couche application transmises entre les entités d'application de communication,

b) de syntaxe de transfert définissant les unités de données du protocole de la couche application transmises entre les entités d'application de communication,

c) de diagrammes d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication, et

d) de diagrammes d'états de relations entre applications définissant le comportement de communication visible entre les entités d'application en communication.

La présente Norme a pour objet de définir le protocole permettant de

a) définir la représentation filaire des primitives de service définies dans la CEI 61158-5-24, et

b) définir le comportement visible de l'extérieur associé à leur transfert.

La présente Norme spécifie le protocole de la couche d'application de bus de terrain de Type 24, conformément au modèle de référence de base OSI (ISO/CEI 7498-1) et à la structure de couche d'application OSI (ISO/CEI 9545).

1.2 Spécifications

L'objet principal de la présente Norme est de spécifier la syntaxe et le comportement du protocole de la couche application qui achemine les services de couche application définis dans la CEI 61158-5-24.

Un objectif secondaire consiste à fournir des voies d'évolution à partir des protocoles de communication industriels antérieurs. C'est ce dernier objectif qui donne lieu à la diversité des protocoles normalisés de la CEI 61158-6.

1.3 Conformité

La présente Norme ne définit pas de mises en œuvre ni de produits particuliers, pas plus qu'elle ne limite les mises en œuvre des entités de couche Application dans les systèmes d'automatisation industriels.

La conformité est assurée par la mise en œuvre de la présente spécification du protocole de la couche application.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

NOTE Toutes les parties de la série CEI 61158, ainsi que la CEI 61784-1 et la CEI 61784-2 font l'objet d'une maintenance simultanée. Les références croisées à ces documents dans le texte se rapportent par conséquent aux éditions datées dans la présente liste de références normatives.

CEI 61158-5-24:2014, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-24 : Définition des services de la couche application – Éléments de type 24*

CEI 61158-6 (toutes les parties), *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6: Spécification du protocole de la couche application*

ISO/IEC 646, *Information technology – ISO 7-bit coded character set for information interchange* (disponible en anglais seulement)

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Partie 1: Le modèle de base*

ISO/CEI 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche Application*

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

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base – Conventions pour la définition des services OSI*

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

ISO/IEC/IEEE 60559:2011, *Information technology – Microprocessor Systems – Floating-Point arithmetic* (disponible en anglais seulement)

3 Termes, définitions, abréviations, symboles et conventions

Pour les besoins du présent document, les termes, définitions, symboles, abréviations et conventions suivants s'appliquent.

3.1 Termes et définitions référencés

Pour les besoins du présent document, les termes, définitions, symboles, abréviations et conventions suivants s'appliquent.

3.1.1 Termes et définitions de l'ISO/CEI 7498-1

Pour les besoins du présent document, les termes suivants définis dans l'ISO/CEI 7498-1 s'appliquent:

a) syntaxe abstraite;
b) entité d'application;
c) processus d'application;
d) unité de données de protocole d'application;
e) invocation de processus d'application;
f) fonctionnalité (N);
g) fonction (N);
h) entités (N) correspondantes;
i) contexte de présentation;
j) système réel;
k) syntaxe de transfert.

3.1.2 Termes et définitions de l'ISO/CEI 9545

Pour les besoins du présent document, les termes suivants définis dans l'ISO/CEI 9545 s'appliquent:

a) association d'applications;
b) contexte d'application;
c) invocation d'entités d'application;
d) type d'entité d'application;
e) élément de service d'application.

3.1.3 Termes et définitions de l'ISO/CEI 8824-1

Pour les besoins du présent document, les termes suivants définis dans l'ISO/CEI 8824-1 s'appliquent:

a) type simple;
b) composant;
c) type de composant;
d) type entier;
e) type bitstring;
f) type octetstring;
g) type null;
h) type séquence;
i) séquence de types;
j) type de choix;
k) type IA5String;
l) codage.

3.1.4 Termes et définitions de l'ISO/CEI 10731

Pour les besoins du présent document, les termes suivants définis dans l'ISO/CEI 10731 s'appliquent:

a) primitive de service OSI; primitive;

b) fournisseur de service OSI; fournisseur;

c) utilisateur de service OSI; utilisateur;

3.1.5 Termes et définitions de l'ISO/CEI 19501

Pour les besoins du présent document, les termes suivants définis dans l'ISO/CEI 19501 s'appliquent:

a) événement;

b) état;

c) diagramme d'états;

d) sous-état;

e) sous-machine;

f) transition.

3.2 Termes et définitions complémentaires

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

3.2.1
alarme
état de l'appareil de terrain signalant qu'il a détecté un problème grave à résoudre et qu'il ne peut pas continuer à fonctionner normalement, par l'intermédiaire du service FDC (commande d'appareil de terrain) du bus de terrain de Type 24

Note 1 à l'article: Les états d'alarme sont verrouillés et il faut les utiliser pour être libérés.

Note 2 à l'article: Les alarmes sont classées en trois groupes: alarmes de communication, alarmes associées à une commande illégale et alarmes spécifiques à l'application. Toutefois, les définitions concrètes dépendent de la mise en œuvre de chaque appareil de terrain.

3.2.2
objet de processus d'application
représentation réseau d'un aspect particulier d'un processus d'application (AP), modélisée sous la forme d'un objet accessible sur le réseau présent dans un AP ou un autre APO

Note 1 à l'article: Voir 9.3.4 de la CEI 61158-1.

3.2.3
contexte de processus d'application
contexte AP
connaissance partagée ou ensemble de règles communes régissant la communication des entités d'application FAL (AE) et décrivant le comportement de communication collectif admissible entre les entités d'application faisant partie intégrante d'un ensemble de relations entre applications (AR)

Note 1 à l'article: Les données d'un contexte AP peuvent être préalablement spécifiées par l'utilisateur, par l'option sélectionnée lors de l'utilisation du service FSM (gestion de bus de terrain) pour lire les fonctionnalités d'un AP homologue, par la fonction de négociation automatique que le système FSM gère, et ainsi de suite. La méthode à adopter dépend de la spécification de chaque mise en œuvre.

3.2.4
type de processus d'application
type AP
description d'une classification des processus d'application (AP) en termes d'ensemble de capacités de la FAL du bus de terrain de Type 24

Note 1 à l'article: Les types AP sont classés en trois catégories: AP du maître C1, AP du maître C2 et AP de l'esclave, en fonction de leurs rôles d'application dans le réseau de bus de terrain.

3.2.5
commande asynchrone
type d'unité de données de protocole d'application (APDU) du service FDC de la FAL de Type 24, qui peut être émis à tout moment à l'issue de la transaction précédente sans tenir compte de la synchronisation avec le cycle de communication

Note 1 à l'article: Les définitions indiquant s'il convient qu'une commande soit de nature asynchrone ou synchrone dépendent d'une application. Elles peuvent être fournies en tant qu'ensemble enregistré de commandes et de réponses ou en tant que profils d'appareil. Voir 4.4 et l'Annexe A.

3.2.6
communication asynchrone
état ou moyen de communication du service FDC de la FAL de Type 24, dans lequel une commande peut être émise à tout moment à l'issue de la transaction précédente sans tenir comte de la synchronisation avec le cycle de communication

Note 1 à l'article: Dans cet état, les commandes synchrones ne peuvent pas être émises, à l'inverse des commandes asynchrones.

3.2.7
attribut
information ou paramètre présent dans les parties variables d'un objet

Note 1 à l'article: En règle générale, ils donnent des informations d'état ou gouvernent le fonctionnement d'un objet. Les attributs peuvent également avoir un impact sur le comportement d'un objet.

3.2.8
maître C1
type d'AP doté des fonctionnalités de maître pour le service FDC de la FAL de Type 24 ou appareil qui met en œuvre ce type d'AP

Note 1 à l'article: Un réseau du bus de terrain de Type 24 ne peut contenir qu'un seul maître C1.

3.2.9
maître C2
type d'AP doté uniquement des fonctionnalités de surveillance pour le service FDC, mais des fonctionnalités du demandeur pour le service de messagerie (MSG) de la FAL de Type 24 ou appareil qui met en œuvre ce type d'AP

Note 1 à l'article: Moins de deux maîtres C2 peuvent exister dans un réseau du bus de terrain de Type 24.

3.2.10
commande
PDU émise par un demandeur ou un maître pour qu'un destinataire ou un esclave exécute certaines fonctions

3.2.11
communication
transfert
transmission
– communication: processus d'échange formel d'informations entre au moins deux appareils, utilisateurs, AP ou entités
– transfert: processus d'acheminement d'une PDU d'un émetteur vers un destinataire
– transmission, processus d'envoi et de propagation de signaux électriques ou de données codées

3.2.12
cycle de communication
période d'activités répétitives synchronisées avec le cycle de transmission pendant l'établissement de la connexion pour le protocole FDC de la FAL de Type 24

Note 1 à l'article: Le cycle de communication peut être synchronisé avec le cycle de transmission multiplié par un facteur d'échelle spécifié.

3.2.13
connexion
contexte ou liaison logique dans des conditions particulières pour le protocole FDC entre un objet maître et un objet esclave de la FAL de Type 24

3.2.14
cyclique
répétitif de manière régulière

3.2.15
communication cyclique
mode de transmission dans lequel les PDU de demande et les PDU de réponse sont échangées de manière répétitive dans les intervalles de temps prévus avec un cycle de transmission pour le protocole de couche inférieure du type 24

Note 1 à l'article: Dans l'AL, le cycle de communication découle du cycle de transmission dans ce mode.

3.2.16
compteur d'échelle de cycle
compteur permettant de générer un cycle de communication par mise à l'échelle d'un cycle primaire ou d'un cycle de transmission

3.2.17
ID d'appareil
partie des "Informations relatives à l'appareil" permettant d'identifier l'appareil pour un type de produit particulier du bus de terrain de Type 24

3.2.18
informations relatives à l'appareil
informations mises en forme et intégrées à l'appareil visant à le caractériser. Il s'agit essentiellement des données d'identification du modèle d'appareil et des paramètres spécifiques au profil de l'appareil pour le bus de terrain de Type 24

3.2.19
profil de l'appareil
ensemble d'informations et de fonctionnalités communes au modèle d'appareil assurant la cohérence entre les différents modèles d'appareil de même type

3.2.20
transfert double
mode de transfert pour le protocole FDC de la FAL de Type 24, dans lequel un émetteur envoie deux fois la même PDU par transaction, que le destinataire utilise pour détecter et corriger une erreur de communication (corruption de données ou perte de données, par exemple) en mode de communication cyclique

3.2.21
communication déclenchée par les événements
mode de transmission du protocole de couche inférieure du bus de terrain de Type 24 dans lequel une transaction commande-réponse-échange se déroule au fur et à mesure des demandes de l'utilisateur

Note 1 à l'article: Le cycle de transmission et le cycle de communication ne surviennent pas dans ce mode.

3.2.22
erreur
condition anormale ou dysfonctionnement de la communication ou de toute autre activité

3.2.23
commande d'appareil de terrain
service FDC
service de communication prioritaire permettant de gérer les données de commande de longueur fixe afin de contrôler un appareil de bus de terrain et les données de réponse correspondantes, avec une limitation rigoureuse quant au délai ou à la gigue pour la temporisation de communication de la FAL de type 24

3.2.24
protocole d'appareil de terrain
protocole FDC
protocole de communication prioritaire permettant de gérer les données de commande de longueur fixe afin de contrôler un appareil de terrain et les données de réponse correspondantes, avec une limitation rigoureuse quant au délai ou à la gigue

3.2.25
maître
classe ou son objet d'instance de l'élément de service d'application (ASE) FDC faisant office de demandeur de commande pour la FAL de Type 24

3.2.26
service de message
service MSG
service de communication qui gère les données de longueur variable et n'impliquant pas de limitation rigoureuse quant au temps de réponse

3.2.27
moniteur (ou surveillant)
classe ou son objet d'instance de l'ASE FDC jouant le rôle d'observateur ou d'abonné des commandes et réponses échangées entre d'autres nœuds de communication pour la FAL de Type 24

3.2.28
esclave de surveillance (ou surveillant)
variante du type d'AP esclave doté d'une classe d'esclave et d'une classe de moniteur pour l'ASE FDC de la FAL de Type 24

3.2.29
horloge réseau
compteur synchronisé et fonctionnant de manière périodique dont chaque nœud du réseau est doté, et qui devient une source d'oscillation du cycle de transmission

3.2.30
cycle primaire
période d'activités répétitives synchronisées avec le cycle de transmission avant l'établissement de la connexion pour le protocole FDC de la FAL de Type 24

3.2.31
machine protocolaire
diagramme d'états qui réalise un protocole comme la fonction principale de l'entité dans chaque couche

3.2.32
demandeur
classe ou son objet d'instance de l'ASE MSG faisant office de demandeur ou d'émetteur de commande pour la FAL de Type 24

3.2.33
répondeur
classe ou son objet d'instance de l'ASE MSG faisant office de répondeur ou de destinataire de la commande pour la FAL de Type 24

3.2.34
réponse
PDU émise par un répondeur ou un esclave afin de donner un résultat ou de signaler l'état de la commande reçue à un demandeur ou un maître

3.2.35
service
opération ou processus qu'un objet exécute à la demande d'un autre objet

3.2.36
transfert simple
mode de transfert normal pour le protocole FDC de la FAL de Type 24 dans lequel un émetteur envoie une seule PDU par transaction

3.2.37
AP esclave
type d'AP doté des fonctionnalités d'esclave pour le service FDC de la FAL de Type 24 ou appareil qui met en œuvre ce type d'AP

3.2.38
esclave
classe ou son objet d'instance de l'ASE FDC faisant office de répondeur pour la FAL de Type 24

3.2.39
commande synchrone
type d'APDU de commande du service FDC de la FAL de Type 24, émis au temps de synchronisation avec chaque cycle de communication

Note 1 à l'article: Les définitions, que la commande soit synchrone ou pas, dépendent d'une application. Elles peuvent être fournies en tant qu'ensemble enregistré de commandes et de réponses ou en tant que profils d'appareil. Voir 4.4 et l'Annexe A.

3.2.40
communication synchrone
état ou moyen de communication pour le service FDC de la FAL de Type 24, dans lequel une commande est émise au temps de synchronisation avec chaque cycle de communication

Note 1 à l'article: Dans cet état, les commandes synchrones et asynchrones peuvent être émises.

Note 2 à l'article: Dans cet état, une erreur de désynchronisation des AP doit être détectée par les mesures du compteur de chien de garde.

3.2.41
cycle de transmission
période d'activités répétitives pour les couches inférieures du bus de terrain de Type 24, dont tous les appareils esclaves sont synchronisés avec celles d'un appareil de maître C1 par le protocole de couche inférieure

3.2.42
mode de transmission
état ou moyen de transmission pour le protocole de couche inférieure du bus de terrain de Type 24; mode cyclique, mode événementiel

3.2.43
espace mémoire virtuel
large bloc de données d'APO pour la FAL de Type 24 qui peut être lu et écrit à l'aide de pseudo adresses mémoires afin d'assurer la cohérence entre les différents modèles d'appareil

Note 1 à l'article: L'espace mémoire virtuel contient les informations relatives à l'appareil et d'autres zones spécifiques au fournisseur. Voir Annexe B.

3.2.44
avertissement
état de l'appareil de terrain signalant qu'il a détecté un problème mineur ou passager, mais qu'il fonctionne toujours normalement, par l'intermédiaire du service FDC (commande d'appareil de terrain) du bus de terrain de Type 24

Note 1 à l'article: Les états d'avertissement sont verrouillés et il est nécessaire de les utiliser pour être libérés.

Note 2 à l'article: Les avertissements sont classés en trois groupes: avertissements de communication, avertissements associés à une commande illégale et avertissements spécifiques à l'application. Toutefois, les définitions concrètes dépendent d'une mise en œuvre de chaque appareil de terrain.

3.3 Abréviations et symboles

Pour les besoins du présent document, les abréviations et symboles suivants s'appliquent.

AE	Application Entity (Entité d'application)
AL	Application Layer (Couche d'application)
A-, AL-	Application Layer (Couche d'application) (préfixe)
ALME	Application Layer Management Entity (Entité de gestion de couche d'application)
AP	Application Process (Processus d'application)
APDU	Application Protocol Data Unit (Unité de données de protocole d'application)
API	Application Process Invocation (Invocation de processus d'application)
APO	Application Process Object (Objet de processus d'application)
APC	Application Process Context (Contexte de processus d'application) (préfixe d'un protocole pour le bus de terrain de Type 24)
APC SM	Application Process Context State Machine (Diagramme d'états du contexte de processus d'application) (pour le bus de terrain de Type 24)
AR	Application Relationship (Relation entre applications)
ASE AR	Application Relationship Application Service Element (Élément de service d'application de relation entre applications)
AREP	Application Relationship End Point (Point final de relation entre applications)
ARPM	Application Relationship Protocol Machine (Machine protocolaire de relation entre applications) (pour le bus de terrain de Type 24)
ARPM-FDCM	ARPM for Field Device Control service Master (ARPM du Maître de service de commande de l'appareil de terrain) (pour le bus de terrain de Type 24)
ARPM-FDCMN	ARPM for Field Device Control service Monitor (ARPM du Moniteur de service de commande de l'appareil de terrain) (pour le bus de terrain de Type 24)
ARPM-FDCS	ARPM for Field Device Control service Slave (ARPM de l'esclave de service de commande de l'appareil de terrain) (pour le bus de terrain de Type 24)
ARPM-MSG	ARPM for Message service (ARPM du service de messagerie) (pour le bus de terrain de Type 24)
ASCII	American Standard Code for Information Interchange (Code normalisé américain pour l'échange d'informations)
ASDU	Application Service Data Unit (Unité de données de service d'application)
ASE	Application Service Element (Élément de service d'application)
ASN.1	Abstract Syntax Notation number One (Notation de syntaxe abstraite numéro un)
CMD	PDU de commande du service FDC (pour le bus de terrain de Type 24)
Cnf	confirm primitive (primitive de confirmation)
DL-	Data Link- (de liaison de données) (préfixe)
DLL	Data Link Layer (Couche liaison de données)
DLM	Data Link-management (Gestion de liaison de données)
DLPDU	Data Link Protocol Data Unit (Unité de données de protocole de liaison de données)
DLSAP	Data link Service Access Point (Point d'accès au service liaison de données)
DLSDU	Data-link Service data unit (Unité de données de service de liaison de données)

DMPM	Data Link Mapping Protocol Machine (Machine protocolaire de mapping de liaison de données)
E2PROM	Electrically erasable Programmable Read Only Memory (Mémoire morte effaçable et programmable électriquement)
FAL	Fieldbus Application Layer (Couche Application de bus de terrain)
FCS	Frame Check Sequence (Séquence de contrôle de trame)
FDC-	Field Device Control (Commande d'appareil de terrain) (préfixe d'un service ou d'un protocole pour le bus de terrain de Type 24)
FDC ASE	Field Device Control Application Service Element (Élément de service d'application de commande d'appareil de terrain) (pour le bus de terrain de Type 24)
FDCPM	Field Device Control Protocol Machine (Machine protocolaire de commande d'appareil de terrain) (pour le bus de terrain de Type 24)
FDCPM-M	Field Device Control Protocol Machine for Master (Machine protocolaire de commande d'appareil de terrain pour le maître) (pour le bus de terrain de Type 24)
FDCPM-MN	Field Device Control Protocol Machine for Monitor (Machine protocolaire de commande d'appareil de terrain pour le moniteur) (pour le bus de terrain de Type 24)
FDCPM-S	Field Device Control Protocol Machine for Slave (Machine protocolaire de commande d'appareil de terrain pour l'esclave) (pour le bus de terrain de Type 24)
FIFO	First In First Out (Premier entré, premier sorti)
FSPM	FAL service protocol machine (Machine protocolaire de service FAL)
FSM-	Fieldbus System Management (Gestion de système de bus de terrain) (préfixe d'un service pour le bus de terrain de Type 24)
FSM ASE	Fieldbus System Management Application Service Element (Élément de service d'application de gestion de système de bus de terrain) pour le bus de terrain de Type 24
IHM	Interface homme-machine
E/S	Entrée/Sortie
ID	Identifiant
Ind	primitive d'indication
LME	Layer Management Entity (Entité de gestion de couche)
Lsb	least significant bit (bit de poids faible)
MAC	Media Access Control (Contrôle d'accès au support)
Msb	most significant bit (bit de poids fort)
MSG	Message (préfixe d'un service ou d'un protocole pour le bus de terrain de Type 24)
MSG ASE	Message Application Service Element (Élément de service d'application de message) pour le bus de terrain de Type 24
MSGPM	Message Protocol Machine (Machine protocolaire de messagerie) pour le bus de terrain de Type 24
MSGPM-RQ	MSGPM du demandeur pour le bus de terrain de Type 24
MSGPM-RS	MSGPM du répondeur pour le bus de terrain de Type 24
OSI	Open Systems Interconnection (Interconnexion des systèmes ouverts)
PM	Protocol machine (Machine protocolaire)
PDU	Protocol Data Unit (Unité de données de protocole)
PhL	Ph-layer (Couche Ph)
QoS	Quality of Service (Qualité de service)
RAM	Random Access Memory (Mémoire vive)
Req	request primitive (primitive de demande)
Rsp	response primitive (primitive de réponse)
RSP	PDU de réponse du service FDC (pour le bus de terrain de Type 24)
SAP	Service Access Point (Point d'accès au service)
SDN	Send Data with no Acknowledge (Transmission de données sans acquittement)
SDU	Service Data Unit (Unité de données de service)
SM	State Machine (Diagramme d'états)
SMIB	System Management Information Base (Base d'informations de gestion système)
UML	Unified Modelling Language (Langage de modélisation unifié)

3.4 Conventions

3.4.1 Conventions générales

La couche Application de bus de terrain (FAL) se compose d'un ensemble d'éléments ASE orientés objet. Chaque ASE est spécifié dans un paragraphe distinct. Chaque spécification d'élément ASE est divisée en trois parties: ses définitions de classe, ses services et la

spécification de ses protocoles. Les deux premières parties sont présentées dans la CEI 61158-5-24. La spécification de protocole de chaque ASE est définie dans la présente Norme.

Les définitions de classe définissent les attributs des classes prises en charge par chaque élément ASE. Dans la présente Norme, chaque attribut est supposé être accessible uniquement à partir de l'objet d'instance du propriétaire lui-même, directement ou grâce aux services de la classe.

La présente Norme emploie les conventions de description énoncées dans l'ISO/CEI 10731.

3.4.2 Conventions du type de données PDU

Les types de données des FAL PDU sont définis avec les notations d'ASN.1.

Les chaînes de caractères dont le premier caractère est "_" (trait de soulignement) font office de symboles de type de données des FAL PDU ou de leurs champs. Une même chaîne de type PDU mais sans le premier caractère "_" indique une instance de PDU du type de données correspondant.

Exemple: CMD-PDU indique une instance de _CMD-PDU et une instruction suivante est ignorée.

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

3.4.3 Conventions dans les diagrammes d'états

Les séquences de protocole sont décrites au moyen de diagrammes d'états.

Dans les schémas d'état, les états sont représentés par des cases et les transitions d'état par des flèches. Leurs conventions sont présentées dans l'ISO/CEI 19501 au format UML (Universal Modeling Language).

Les noms d'états et de transitions du schéma d'état correspondent aux noms indiqués dans la liste des transitions d'état.

Cette liste de transitions d'état est structurée comme indiqué au Tableau 1.

La première ligne contient le nom de la transition par un numéro d'index.

La deuxième ligne définit l'état source ou l'état en cours avant la transition.

La troisième ligne contient un événement et des actions. L'événement est suivi d'arguments facultatifs entre parenthèses, "(" et ")", et de conditions de garde entre crochets, "[" et "]", comme la première ligne. Et les actions commençant par le caractère "/" suivent la ligne d'événement.

La dernière ligne contient l'état cible ou l'état suivant après la transition.

Si l'événement se produit et que les conditions sont remplies, la transition se lance, c'est-à-dire que les actions sont exécutées et il y a une entrée dans le prochain état.

Tableau 1 – Descriptions de transition d'état

T#	État source	Événement (arguments) [conditions] /Action	État cible

Une signification de chaque élément du Tableau 1 est présentée au Tableau 2, selon la définition de "transition" UML (ISO/CEI 19501).

Tableau 2 – Description des éléments de diagramme d'états

Élément de description	Signification
État source État cible	Noms de l'état d'origine et de l'état cible de la transition.
T#	Nom ou numéro de la transition.
Événement	Nom ou description de l'événement déclencheur de la transition.
(arguments)	Valeur de paramètre, expression ou séquence d'éléments séparés par "," pour l'événement. La parenthèse ouvrante "(" et la parenthèse fermante ")" ne font pas partie de la liste d'arguments.
[conditions]	Expression booléenne, qui est vraie pour la transition à lancer. Le crochet ouvrant ("[") et le crochet fermant ("]") ne font pas partie de la condition.
/Action	Liste des affectations et des appels de service ou de fonction. Il convient que l'action soit atomique. La barre oblique "/" qui précède ne fait pas partie de l'action.

NOTE "(arguments)" et "[condition]" peuvent être ignorés s'ils s'avèrent inutiles.

Les conventions utilisées dans les diagrammes d'états sont présentées au Tableau 3.

Tableau 3 – Conventions utilisées dans les diagrammes d'états

Convention	Signification
/*-- description --*/	Décrit et explique les conditions et/ou la procédure à l'aide d'une phrase normale placée entre "/*--" et "--*/" au lieu d'utiliser la notation de pseudo-code. Exemple: /*-- La PM examine une occurrence d'alarmes pour l'appareil distant correspondant. -- Si une alarme est détectée, la PM le signale au FSM ASE. -- Dans le cas contraire, la PM transfère MSGService-PDU à la DLL. --*/
=	La valeur d'une entité de gauche est remplacée par celle d'une entité de droite. Si une entité de droite est un paramètre, il provient de la primitive identifiée comme un événement d'entrée.
Axx	Nom de paramètre si "a" est une lettre Exemple: Identifier = reason; signifie que la valeur d'un paramètre 'reason' est attribuée à un paramètre appelé 'Identifier.'
"xxx"	Chaîne visible fixe Exemple: Identifier = "abc"; signifie que la valeur "abc" est attribuée à un paramètre nommé 'Identifier'
Nnn	Si tous les éléments sont des chiffres, l'élément représente une constante numérique notée en représentation décimale.
0xnn	Si tous les éléments nn sont des chiffres, l'élément représente une constante numérique notée en représentation hexadécimale.
==	Condition logique indiquant qu'un élément de gauche est égal à un élément de droite.
<	Condition logique indiquant qu'un élément de gauche est inférieur à un élément de droite.
>	Condition logique indiquant qu'un élément de gauche est supérieur à un élément de droite.

Convention	Signification
!=	Condition logique indiquant qu'un élément de gauche n'est pas égal à un élément de droite.
&&	"AND" (c'est-à-dire: ET) logique
||	"OR" (c'est-à-dire: OU) logique
!	"NOT" (c'est-à-dire: PAS) logique
+ - * /	Opérateurs arithmétiques
;	Séparateur d'expressions

D'autres notations définies en langage C (ISO/CEI 9899) peuvent être utilisées pour décrire les conditions et les actions.

4 Syntaxe abstraite

4.1 Types de données de base

Les types de données suivants peuvent être utilisés dans la FAL de Type 24.

a) Types de base (types simples d'ASN.1):

i) INTEGER;

ii) REAL;

iii) BIT STRING;

iv) OCTET STRING;

v) NULL.

b) Types de base spécifiques (sous-types issus de l'ASN.1):

vi) Integer8 ::= INTEGER (-128..127);

vii) Integer16 ::= INTEGER (-32 768 .. 32 767);

viii) Integer32 ::= INTEGER (-2 147 483 648 .. 2 147 483 647);

ix) Integer64 ::= INTEGER ((-1) × ‘8000 0000 0000 0000’H..‘7FFF FFFF FFFF FFFF’H);

x) Unsigned8 ::= INTEGER (0..’FF’H);

xi) Unsigned16 ::= INTEGER (0..’FFFF’H);

xii) Unsigned32 ::= INTEGER (0..’FFFF FFFF’H);

xiii) Unsigned64 ::= INTEGER (0..’FFFF FFFF FFFF FFFF’H);

xiv) Float32 ::= 0 | infinie négative | infinie positive

| REAL (WITH COMPONENTS { mantisse ((-1)× c_{32}), base (2), exposant (q_{32}) })

| REAL (WITH COMPONENTS { mantisse (c_{32}), base (2), exposant (q_{32}) })

avec,

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

q_{32} ::= INTEGER (-127..127).

NOTE 1 La plage de valeurs réelles est comprise entre l'infinie négative et l'infinie positive. Mais la précision représentable est conforme à binary32 ou la précision seule à l'ISO/CEI/IEEE 60559. Voir 5.2.2.

xv) Float64 ::= 0 | infinie négative | infinie positive

| REAL (WITH COMPONENTS { mantisse ($(-1)\times c_{64}$), base (2), exposant (q_{64}) })

| REAL (WITH COMPONENTS { mantisse (c_{64}), base (2), exposant (q_{64}) })

avec,

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

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

NOTE 2 La plage de valeurs réelles est comprise entre l'infinie négative et l'infinie positive. Mais la précision représentable est conforme à binary64 ou l la précision double à l' ISO/CEI/IEEE 60559. Voir 5.2.2.

xvi) IA5String

c) Types de structure avec types de composant ASN.1:

Le type Structure est défini par une combinaison de types SEQUENCE, de types CHOICE d'ASN.1, de types de base et d'éléments de sous-type définis dans l'ASN.1. La FAL PDU décrite en 4.2 et 4.3 correspond au type Structure.

xvii) SEQUENCE

xviii) CHOICE

d) Types de matrice avec un type de composant ASN.1:

Les types de matrice sont une liste de types de données définis avec le type SEQUENCE OF de l'ASN.1. Les types de matrice classiques sont présentés avec le type SEQUENCE OF ci-dessous:

xix) SEQUENCE OF;

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

xxi) Array8 ::= SEQUENCE OF Integer8;

xxii) Array16 ::= SEQUENCE OF Integer16;

xxiii) Array32 ::= SEQUENCE OF Integer32;

xxiv) Array64 ::= SEQUENCE OF Integer64.

4.2 Types FAL PDU

4.2.1 Plans de types APDU: _APDU

Le type APDU: _APDU doit être composé de deux groupes: _FDCServicePDU et _MSGServicePDU.

_FDCServicePDU doit être composé d'une paire de PDU. _CMD-PDU et _RSP-PDU sont respectivement utilisés pour les commandes FDC et ses réponses.

De plus _MSGServicePDU doit être composé d'une paire de PDU. _MSGREQ-PDU et _MSGRSP-PDU sont respectivement utilisés pour les demandes MSG et ses réponses.

```
_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
```

La Figure 1 présente des définitions de type d'APDU par la structure arborescente.

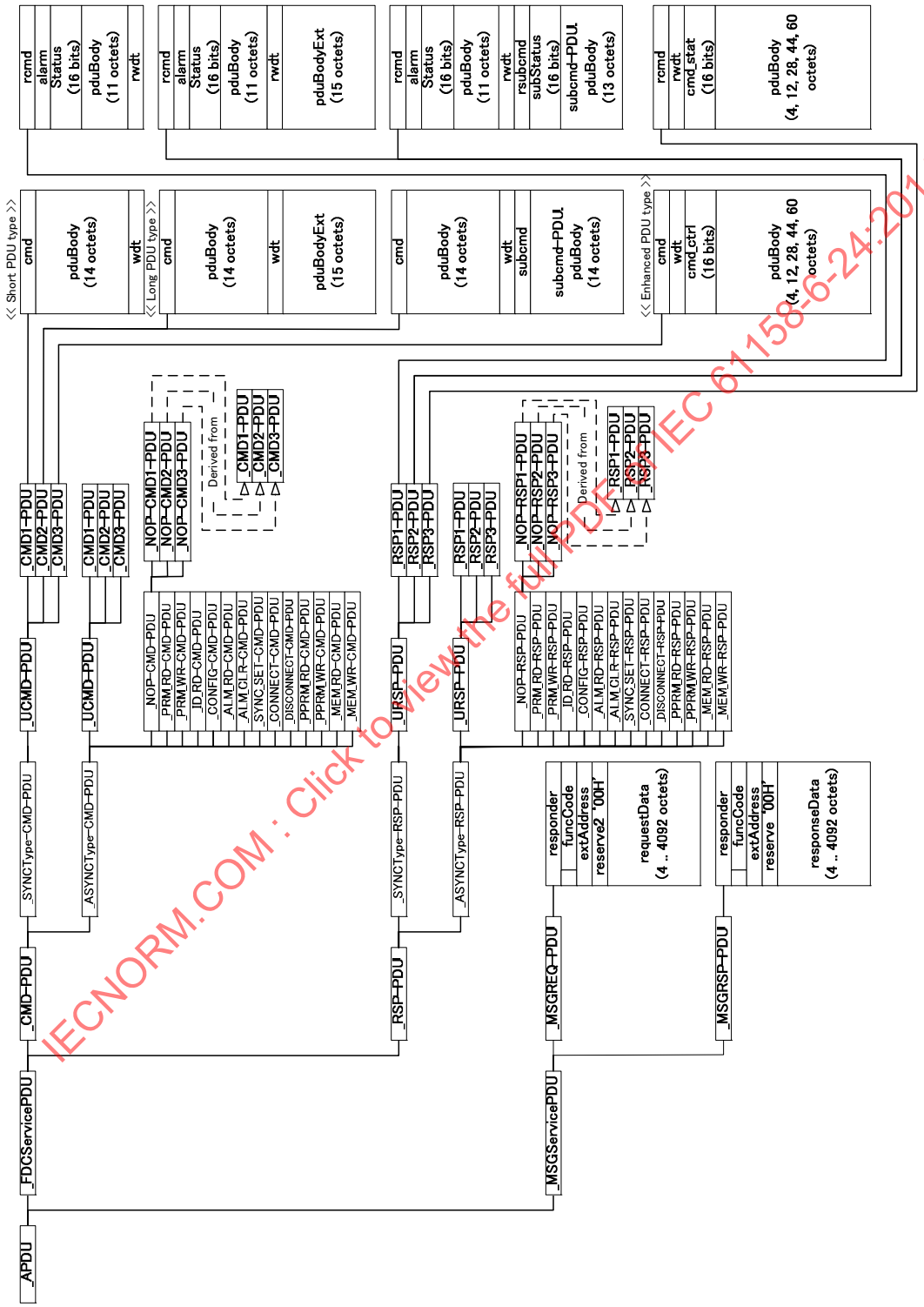


Figure 1 – Structure arborescente des types APDU

4.2.2 PDU pour le service de commande de l'appareil de terrain

4.2.2.1 Présentation

FDCServicePDU doit être édité avec une SDU dans la primitive de service FDC et être envoyé par l'intermédiaire de l'AR. Un FDC ASE reçoit la PDU de l'ASE homologue par l'intermédiaire de l'AR.

Le type de PDU de la commande FDC: _CMD_PDU doit être classé dans la commande de type sync (commande de synchronisation): _SYNCType-CMD-PDU et commande de type async (commande asynchrone): _ASYNCType-CMD-PDU. Les deux types sont des formats PDU identiques: _UCMD-PDU. Par conséquent, la FDC PM ne peut pas distinguer les PDU de leurs types. Il convient que l'utilisateur du FDC ASE sache quelles commandes appartiennent aux commandes synchrones ou celles qui ne lui appartiennent pas, et puisse distinguer les types à l'aide du champ cmd en tant que code clé dans l'en-tête PDU.

PDU de commande asynchrone: _ASYNCType-CMD_PDU doit inclure _UCMD-PDU et quatorze autres types de PDU en tant que commandes communes, qui sont également des variantes de _UCMD-PDU.

Un ensemble de commandes et de réponses, déduit de _UCMD-PDU et de _URSP-PDU présentés ci-après, peut être défini pour une application particulière (commande d'entraînement moteur, servo-commande, etc.) par les utilisateurs FAL. Cet ensemble peut être enregistré en tant que profil d'appareil (voir 4.4 et Annexe A). De plus, un utilisateur FAL peut choisir les profils d'appareil pris en charge par un appareil de terrain cible, lorsqu'une connexion est établie par le service FDC-Connect (voir 8.2.4.3 et 6.4.1.2.3.4 de la CEI 61158-5-24).

```
_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
_UCMD-PDU               ::= _CMD1-PDU | _CMD2-PDU | _CMD3-PDU
_NOP-CMD-PDU            ::= _NOP-CMD1-PDU | _NOP-CMD2-PDU | _NOP-CMD3-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
```

Le type de PDU de réponse FDC: _RSP_PDU doit également être classé dans la commande de type sync (réponse de synchronisation): _SYNCType-RSP-PDU et commande de type async (réponse asynchrone): _ASYNCType-RSP-PDU. Les deux types doivent être déduits de formats PDU identiques: _URSP-PDU. Par conséquent, la FDC PM ne peut pas distinguer les PDU de leurs types. Il convient que l'utilisateur du FDC ASE sache quelles réponses appartiennent aux réponses synchrones ou celles qui ne lui appartiennent pas, et puisse distinguer les types à l'aide du champ rcmd en tant que code clé dans l'en-tête PDU. De plus, le code de préfixe d'une PDU de réponse correspond à celui d'une PDU de commande, une réponse devant être échangée contre une commande.

Le type de PDU de réponse asynchrone: _ASYNCType-RSP_PDU doit inclure _URSP-PDU et quatorze autres types de PDU en tant que réponses communes à une application, qui sont également des variantes de _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
_URSP-PDU               ::= _RSP1-PDU | _RSP2-PDU | _RSP3-PDU
_NOP-RSP-PDU            ::= _NOP-RSP1-PDU | _NOP-RSP2-PDU | _NOP-RSP3-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
```

4.2.2.2 Composant commun de la PDU

4.2.2.2.1 PDU de commande FDC

_UCMD-PDU peut comporter trois types: le type abrégé _CMD1-PDU, le type long _CMD2-PDU et le type amélioré _CMD3-PDU. _CMD1-PDU et _CMD2-PDU doivent comporter les mêmes champs d'en-tête, mais celui de _CMD3-PDU doit être amélioré.

La longueur pduBody de _CMD1-PDU doit être de 14 octets lorsqu'il est codé. Il en est de même pour _CMD2-PDU. Cette partie est appelée commande principale, et _CMD2-PDU doit également comporter un autre champ pduBodyExt étendu, et peut superposer une PDU de commande supplémentaire appelée sous-commande: _SUBCMD-PDU sur ce champ.

La longueur pduBody de _CMD3-PDU doit pouvoir être sélectionnée entre 4, 12, 28, 44, et 60 octets, mais elle peut ne pas être commutée à la volée.

```
_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
                                                                    type
                        cmd         _CMDCode,
                        wdt         _WDT,
                        cmd_ctrl    _CMD_CTRL,
                        pduBody     OCTET STRING SIZE (4|12|28|44|60) }
```

a) champ cmd

cmd
champ de données contenant le code d'identification du contenu de chaque commande

Ce champ doit être codé en _CMDCode, sous-type d'Unsigned8, avec la plage de valeurs ci-dessous. Les commandes communes prédéfinies sont présentées en tant que _PrimaryCMD ci-dessous.

– '00'H à '1F'H: utilisé ou réservé pour les commandes communes;
– '20'H à 'BF'H: réservé aux commandes spécifiques à l'application;
– 'C0'H à 'FF'H: réservé aux fournisseurs.

```
_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) }
```

b) champ wdt

wdt
champ du compteur d'horloge de surveillance

Ce champ doit être utilisé pour permettre à l'esclave de détecter l'activité non synchrone ou l'erreur WDT du maître correspondant. Le maître doit compter le champ wdt des PDU d'envoi à chaque cycle de communication en cas d'état de communication synchrone. De plus, l'esclave doit examiner la valeur du champ des PDU destinataires à chaque cycle de communication pour détecter l'erreur wdt.

Le type de données est _WDT. Il doit comporter deux sous-champs: mn et sn.

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

mn (valeur de comptage du maître)

La plage de valeurs doit être comprise entre 0 et 15.

Le champ mn d'une CMD-PDU suivante à envoyer doit faire l'objet d'un comptage progressif de un. Un maître FDC doit procéder ainsi à chaque cycle de communication. D'autre part, lorsqu'un esclave FDC reçoit la PDU, il doit examiner le compteur à chaque cycle. Si le compteur ne correspond pas à la dernière valeur mn plus un, il doit alerter le compteur de l'horloge de surveillance de l'erreur.

sn (valeur de comptage de l'esclave)
La plage de valeurs doit être comprise entre 0 et 15.

La même valeur que le champ rsn de la dernière RSP-PDU reçue doit être attribuée au champ sn d'une CMD-PDU suivante à envoyer. Un maître FDC doit la copier depuis la dernière RSP-PDU à chaque cycle de communication.

c) champ subcmd

subcmd
champ de données contenant le code d'identification du contenu de chaque sous-commande

Le type de données est _SubCMD. En principe, la définition des codes des sous-commandes doit être identique à celle du champ cmd principal, mais l'utilisation de certains codes peut être restreinte s'ils rencontrent des difficultés à traiter des commandes parallèles. Les sous-commandes communes prédéfinies sont présentées en tant que _PrimarySubCMD ci-dessous.

```
_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) }
```

d) champ cmd_ctrl

cmd_ctrl
ensemble de bits de contrôle indépendants de toute transaction de commande, entre le maître FDC et l'esclave FDC

Le type de données est _CMD_CTRL. Il doit comporter deux sous-champs significatifs: alm_clr et 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
Ce champ doit être utilisé pour commander la libération de l'état d'alarme/avertissement de l'appareil esclave.

valeur 0: ne rien exécuter

la valeur passe à 1: libérer l'état

cmd_id
Si l'appareil maître adresse en permanence le contenu d'une CMD-PDU à l'esclave, il peut être utilisé pour permettre à l'appareil esclave de reconnaître la commande comme étant une commande différente.

La plage de valeurs doit être comprise entre 0 et 3.

Le maître FDC peut utiliser cmd_id pour permettre à un esclave FDC de reconnaître qu'il s'agit d'une nouvelle commande lorsque le maître envoie la même commande à l'esclave de manière répétée.

Étant donné que l'esclave doit renvoyer l'écho du champ cmd_id de chaque commande par l'intermédiaire du champ RSP-PDU.cmd_stat.rcmd_id présenté ci-après, le maître peut également juger la commande pour laquelle la station esclave a envoyé la réponse.

L'utilisation du champ cmd_id n'est pas obligatoire pour le maître.

Étant donné que RSP-PDU.cmd_stat.cmdRdy = 0 dans les PDU de réponse, l'esclave doit ignorer les commandes dont le champ cmd_id est différent, et poursuivre l'exécution de la commande qui a déjà été acceptée.

4.2.2.2.2 PDU de réponse FDC

_URSP-PDU peut comporter trois types: le type abrégé _RSP1-PDU, le type long _RSP2-PDU et le type amélioré _RSP3-PDU. _RSP1-PDU et _RSP2-PDU doivent comporter des champs d'en-tête communs, mais celui de _RSP3-PDU doit être amélioré.

La longueur pduBody de _RSP-PDU doit être de 11 octets. Il en est de même pour _RSP2-PDU. Il s'agit d'une réponse principale ou d'une réponse de la commande principale, _RSP2-PDU devant également comporter un autre champ pduBodyExt étendu et pouvoir superposer une PDU de commande supplémentaire appelée sous-réponse ou réponse d'une sous-commande: _SUBRS-PDU sur ce champ.

La longueur pduBody de _RSP3-PDU doit pouvoir être sélectionnée entre 4, 12, 28, 44, et 60 octets, mais elle peut ne pas être commutée à la volée.

```
_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
                                                                          PDU type
                                     rsubcmd   _SubCMD,
                                     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) }
```

a) champ rcmd

rcmd
champ de données contenant le même code qu'une commande reçue et de traitement

Ce champ doit donc être codé en _CMDCode. Voir 4.2.2.2.1.

b) champ alarm

alarm
Ce champ doit contenir le code d'alarme.

Le type de données est _ALARM. La plage de valeurs doit être comprise entre 0 et 255.

```
_ALARM                  ::= Unsigned8
```

c) champ status

status
Ce champ doit indiquer l'état de l'appareil esclave.

Le type de données est _STATUS. Il doit comporter trois sous-champs significatifs, comme suit.

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

alarm
Ce champ binaire doit indiquer l'état d'alarme dans l'appareil esclave.

0: Pas d'alarme,

1: Des alarmes se sont produites.

warning
Ce champ binaire doit indiquer l'état d'avertissement dans l'appareil esclave.

0: Pas d'avertissement,

1: Des avertissements se sont produits.

cmdRdy
Ce champ binaire doit indiquer l'état de progression de la commande de l'esclave FDC.

L'appareil esclave doit conserver le bit cmdRdy 0 lors du traitement de la commande. À l'issue du traitement, l'appareil esclave doit remplacer le bit 0 par le bit 1.

Pour signaler l'exécution de la commande spécifique, le maître FDC ne doit pas uniquement examiner le bit, mais également certains autres champs RSP-PDU contenant le champ CMD-PDU correspondant afin de faire la distinction avec la transaction précédente.

Si la durée de rétention de cmdRdy = 0 a expiré, le maître doit générer une erreur de temporisation de commande. La valeur du temporisateur dépend de chaque spécification de produit pour les appareils esclaves.

Une modification de l'état de ce bit doit être indépendante de l'état d'alarme ou

d'avertissement.

0: occupé avec exécution de commande en cours

1: prêt pour la nouvelle commande

d) champ rwdt

rwdt
champ du compteur d'horloge de surveillance

Ce champ doit être utilisé pour permettre au maître de détecter l'activité non synchrone ou l'erreur WDT de l'esclave correspondant. L'esclave doit compter et mettre à jour le champ rwdt des PDU d'envoi à chaque cycle de communication en cas d'état de communication synchrone. De plus, le maître doit examiner la valeur du champ des PDU destinataires à chaque cycle de communication pour détecter l'erreur rwdt.

Le type de données est _RWDT. Il doit comporter deux sous-champs: rmn et rsn.

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

rmn (valeur de comptage du maître)
La plage de valeurs doit être comprise entre 0 et 15.

La même valeur que le champ mn de la dernière CMD-PDU reçue doit être attribuée au champ rmn d'une RSP-PDU suivante à envoyer. Un esclave FDC doit la copier depuis la CMD-PDU à chaque cycle de communication.

rsn (valeur de comptage de l'esclave)
La plage de valeurs doit être comprise entre 0 et 15.

Le champ rsn d'une RSP-PDU suivante à envoyer doit faire l'objet d'un comptage progressif de un. Un esclave FDC doit procéder ainsi à chaque cycle de communication. D'autre part, lorsqu'un maître FDC reçoit la PDU, il doit examiner le compteur à chaque cycle. Si le compteur ne correspond pas à la dernière valeur rsn plus un, il doit alerter le compteur de l'horloge de surveillance de l'erreur.

e) rsubcmd

rsubcmd
champ de données contenant le même code que la sous-commande reçue et de traitement

Ce champ doit être codé en _SubCMD. Voir 4.2.2.2.1.

f) subStatus

subStatus
Ce champ doit indiquer l'état de la station esclave dans la partie PDU de sous-réponse.

Le type de données est _SubSTATUS. Ce champ doit être codé comme le champ status.

```
_SUBSTATUS               ::= _STATUS
```

g) champ cmd_stat

cmd_stat
Ce champ doit indiquer l'état de l'appareil esclave.

Le type de données est _CMD_STAT. Il doit comporter sept sous-champs, comme suit.

```
_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
Ce champ binaire doit indiquer l'état d'occurrence de l'alarme spécifique à l'appareil dans l'appareil esclave.

0: l'esclave n'est pas dans un état d'alarme spécifique à l'appareil;

1: l'esclave est dans un état d'alarme spécifique à l'appareil, sauf comm_alm et cmd_alm.

La spécification de l'état d'alarme de l'appareil peut dépendre de la mise en œuvre du produit spécifiée, mais il convient de classer séparément un facteur d'alarmes d'un problème de communication et de commande.

Après la reprise de l'esclave suite à l'état d'alarme, grâce à l'exécution de la commande ALM_CLR-CMD-PDU ou du bit cmd_ctrl.alm_clr dans une CMD-PDU, ce bit doit être mis à 0 (d_alm = 0)

d_war
Ce champ binaire doit indiquer l'état d'occurrence de l'avertissement spécifique à l'appareil dans la station esclave.

0: l'esclave n'est pas dans un état d'alarme spécifique à l'appareil;

1: l'esclave est dans un état d'alarme spécifique à l'appareil, sauf comm_alm et cmd_alm.

La spécification de l'état d'alarme de l'appareil peut dépendre de la mise en œuvre du produit spécifiée, mais il convient de ne pas classer un facteur d'alarmes parmi les problèmes de communication et de commande.

Après la reprise de l'esclave suite à l'état d'avertissement, grâce à l'exécution de la commande ALM_CLR-CMD-PDU ou du bit cmd_ctrl.alm_clr dans une CMD-PDU, ce bit doit être mis à 0 (d_war = 0)

cmdRdy
Ce champ binaire doit indiquer l'état de progression de la commande de l'esclave FDC.

Ce bit a la même signification que RSP1_PDU.status.cmdRdy.

0: occupé avec exécution de commande en cours;

1: prêt pour la nouvelle commande.

alm_clr_cmp
Ce champ binaire doit indiquer l'état d'exécution du processus de libération d'alarme.

0: Libération d'alarme/avertissement non terminée;

1: Libération d'alarme/avertissement terminée.

rcmd_id
Ce champ doit être un code clé permettant d'indiquer à quelle commande correspond la PDU de réponse, en écho au cmd_id de la PDU de commande correspondante.

La plage de valeurs est comprise entre 0 et 3.

cmd_alm
Ce champ doit notifier un code d'alarme en cas de commande anormale.

La signification de chaque code doit être la suivante:

'00'H: Normal;
'01'H: Avertissement sur données en dehors de la plage;
'02'H à '07'H: réservé aux codes d'avertissement;
'08'H: Alarme sur absence de prise en charge;
'09'H: Alarme sur données en dehors de la plage;
'0A'H: Alarme sur condition d'exécution de commande anormale;
'0B'H: Alarme sur combinaison de sous-commande anormale;
'0C'H: Alarme sur phase anormale;
'0D'H à '0F'H: réservé aux codes d'alarme;

comm_alm
Ce champ doit notifier un code d'alarme pour l'état d'erreur de communication.

La signification de chaque code doit être la suivante:

'00'H: Normal;
'01'H: Avertissement sur FCS anormale;
'02'H: Avertissement sur réception anormale;
'03'H à '07'H: réservé à l'avertissement;
'08'H: Alarme sur FCS anormale;
'09'H: Alarme sur réception anormale;
'0A'H à '0F'H: réservé à l'alarme.

4.2.3 PDU du service de messagerie

```
_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)}
```

a) champ responder

responder
Ce champ doit contenir l'adresse de nœud de destination ou le répondeur du message.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

– '00'H: Réservé;

- '01'H: appareil maître C1;
- '02'H: appareil maître C2;
- '03'H à 'EF'h: Appareil esclave;
- 'F0'H à 'FF'H: réservé.

b) champ funcCode

funcCode
Ce champ doit contenir le code indiquant la fonction du message.

La plage de codes doit être comprise entre '00'H et '7F'H.

c) champ errorFlag

errorFlag
Ce champ doit indiquer l'état de réponse du message. La valeur 0 doit être attribuée pour une réponse normale et la valeur 1 pour une réponse anormale.

- 0: réponse normale;
- 1: réponse erronée.

d) champ extAddress

extAddress
Ce champ doit contenir la sous-adresse représentant un sous-appareil lorsque le nœud de destination est un appareil intégré à au moins deux sous-appareils.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs comprise entre '00'H et 'FE'H. Le code 'FF'H est réservé.

e) champs requestData et respondeData

requestData et responseData
Ces champs doivent contenir les SDU qui peuvent être définies par l'utilisateur pour chaque code de fonction. Un exemple d'ensemble de commandes de message utilisateur pour le code de fonction est présenté ci-après: '42'H est présenté à l'Annexe C.

4.3 Définitions détaillées des _FDCService-PDU

4.3.1 Type de PDU abrégé

4.3.1.1 Commande et réponse NOP

NOP symbolise une commande "No OPeration" (pas d'opération). À la réception de cette commande, l'esclave ne doit rien faire, sauf répondre à une commande NOP-RSP1-PDU indiquant les derniers état et alarme.

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

4.3.1.2 Commande et réponse PRM_RD

PRM_RD symbolise une commande "parameter-read" (lecture de paramètre). À la réception de cette commande, l'esclave doit lire une valeur de paramètre spécifiée dans le champ PRM_RD-RSP1-PDU.parameter et doit y répondre

```
_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) }
```

a) champ pNo

pNo
Ce champ doit contenir le nombre de paramètres à lire.

La plage de valeurs doit être comprise entre 0 et 65 535.

b) champ pSize

pSize
Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit être comprise entre 0 et 255.

c) champ parameter

parameter
Ce champ doit contenir les données lues correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.3.1.3 Commande et réponse PRM_WR

PRM_WR symbolise une commande "parameter-write" (écriture de paramètre). A la réception de cette commande, l'esclave doit écrire une valeur de paramètre spécifiée dans le champ PRM_WR-CMD1-PDU.parameter et doit répondre en écho pour signaler l'exécution de la commande.

```
_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) }
```

a) champ pNo

pNo
Ce champ doit contenir le nombre de paramètres à écrire.

La plage de valeurs doit être comprise entre 0 et 65 535.

b) champ pSize

pSize
Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit être comprise entre 0 et 255.

c) champ parameter

parameter
Ce champ doit contenir les données à écrire correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.3.1.4 Commande ID_RD et réponse

ID_RD symbolise une commande "device-ID-read" (lecture d'ID d'appareil). A la réception de cette commande, l'esclave doit lire une partie d'un élément spécifié dans les informations relatives à l'appareil dans le champ ID_RD-RSP1-PDU.idData et doit y répondre. La sémantique et les types de données des informations relatives à l'appareil peuvent dépendre du modèle d'appareil, sauf idCode: 0, qui doit spécifier les informations relatives au modèle d'appareil.

```
_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) }
```

a) champ idCode

idCode
Ce champ doit contenir le code d'ID d'appareil à lire.

La plage de valeurs doit être comprise entre 0 et 255.

- '00'H: modèle d'appareil;
- '01'H à 'FF'H: réservé.

b) champ idOffset

idOffset
Ce champ doit contenir l'adresse de décalage de l'ID d'appareil à lire.

La plage de valeurs doit être comprise entre 0 et 255.

c) champ idSize

idSize
Ce champ doit contenir la taille de lecture en octets de l'ID d'appareil.

La plage de valeurs doit être comprise entre 0 et 255.

d) champ idData

idData
Ce champ doit contenir les données lues correspondant au champ idCode. La taille des données, la structure des données et leur signification peuvent dépendre du champ idCode.

4.3.1.5 Commande CONFIG et réponse

CONFIG symbolise une commande "configure-device" (configurer appareil). À la réception de cette commande, l'esclave doit activer un ensemble de paramètres sur la RAM mise à jour avec une commande PAR_WR, puis doit répondre pour signaler l'exécution de la commande.

```
_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)} )
```

a) champ config_mode

config_mode
Ce champ doit contenir le mode de configuration de l'appareil.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

– pActive ('00'H): pour procéder à un nouveau calcul avec les paramètres et la configuration;
– '01'H à 'FF'H: réservé.

4.3.1.6 Commande ALM_RD et réponse

ALM_RD symbolise une commande "alarm-read" (lecture d'alarme). A la réception de cette commande, l'esclave doit lire un ensemble de données d'alarme comprenant les détails de l'état d'alarme et d'avertissement dans le champ ALM_RD-RSP1-PDU.alm_data, puis il doit répondre avec ces éléments. La sémantique et les types de données du champ alm_data peuvent dépendre du modèle d'appareil.

```
_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) }
```

a) champ alm_rd_mode

alm_rd_mode
Ce champ doit contenir le mode de lecture d'alarme.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

– currentAlm ('00'H): lire l'état d'alarme/avertissement en cours;
– 01'H à 'FF'H: réservé.

b) champ alm_data

alm_data
Ce champ doit contenir l'état d'alarme lu correspondant au champ alm_rd_mode.

4.3.1.7 Commande et réponse ALM_CLR

ALM_CLR symbolise une commande "alarm-clear" (annulation d'alarme). A la réception de cette commande, l'esclave doit annuler l'état d'alarme et d'avertissement interne, puis il doit répondre pour signaler l'exécution de la commande.

Avec cette commande, on ne peut pas remédier à la cause de l'alarme ou de l'avertissement. Il convient de l'utiliser après avoir résolu les causes ou problèmes réels afin de rétablir l'état normal de l'appareil.

```
_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) }
```

a) champ alm_clr_mode

alm_clr_mode
Ce champ doit contenir le mode d'annulation d'alarme.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

– cAlmClear ('00'H): annuler l'état d'alarme/avertissement en cours;
– '01'H à 'FF'H: réservé.

4.3.1.8 Commande SYNC_SET et réponse

SYNC_SET symbolise une commande "set into synchronous-state" (définir à l'état synchrone). À la réception de cette commande, l'esclave doit basculer la PM esclave (FDCPM-S) de l'état S2: AsyncConnected à l'état S3: SyncConnected. À la réception de la réponse, le maître doit, de la même manière, basculer la PM maître (FDCPM-M) de l'état S2: AsyncConnected à l'état S3: SyncConnected. Voir 8.2.4 pour plus de détails.

```
_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 Commande CONNECT et réponse

Dans le cadre de l'échange de cette commande et de la réponse, le maître et l'esclave doivent établir une connexion FDC AR. À cet instant, certaines options ou certains paramètres de communication peuvent être spécifiés avec les PDU. Ils doivent assurer la transition de chaque PM (FDCPM-M et FDCPM-S). Voir 8.2.4 et 8.2.5 pour plus de détails.

```
_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)}
```

a) champ com_mod

com_mod
Ce champ doit contenir plusieurs modes relatifs à la communication dans la connexion.

syncmode
Ce champ binaire doit indiquer le mode de communication.

0: état de communication asynchrone;

1: état de communication synchrone.

dtmode
Ce champ binaire doit indiquer le mode de transfert.

Ce champ doit contenir la plage de valeurs ci-dessous.

'00'H: mode de transfert unique;

'01'H: mode de transfert double;

02'H à '03'H: réservé.

subcmd
Ce champ binaire doit préciser l'utilisation ou pas de la sous-commande.

0: Champ de sous-commande désactivé;

1: Champ de sous-commande activé.

b) champ com_time

com_time
Ce champ doit contenir la durée de cycle de communication sous la forme d'un multiple de la durée de cycle de transmission.

La plage de valeurs doit être comprise entre 0 et 255.

4.3.1.10 Commande DISCONNECT et réponse

Dans le cadre de l'échange de cette commande et de la réponse, le maître et l'esclave doivent libérer une connexion FDC AR et les options de communication spécifiées doivent être réinitialisées. Ils doivent assurer la transition de chaque PM (FDCPM-M et FDCPM-S). Voir 8.2.4 et 8.2.5 pour plus de détails.

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

4.3.1.11 Commande PPRM_RD et réponse

PRM_RD symbolise une commande "PROM-parameter-read" (lecture de paramètre PROM). À la réception de cette commande, l'esclave doit lire un paramètre spécifié sur E2PROM ou dans la mémoire non volatile dans le champ PPRM_RD-RSP1-PDU.parameter et doit y répondre

```
_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) }
```

a) champ pNo

pNo
Ce champ doit contenir le nombre de paramètres à lire.

La plage de valeurs doit être comprise entre 0 et 65 535.

b) champ pSize

pSize
Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit être comprise entre 0 et 255.

c) champ parameter

parameter
Ce champ doit contenir les données correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.3.1.12 Commande PPRM_WR et réponse

PRM_WR symbolise une commande "PROM-parameter-write" (lecture de paramètre PROM). À la réception de cette commande, l'esclave doit écrire une valeur de paramètre spécifiée dans le champ PPRM_WR-CMD1-PDU.parameter dans la mémoire E2PROM ou la mémoire non volatile, puis il doit répondre en écho pour signaler l'exécution de la commande.

```
_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) }
```

a) champ pNo

pNo
Ce champ doit contenir le nombre de paramètres à écrire.

La plage de valeurs doit être comprise entre 0 et 65 535.

b) champ pSize

pSize
Ce champ doit contenir la taille du paramètre en octets.

La plage de valeurs doit être comprise entre 0 et 255.

c) champ parameter

parameter
Ce champ doit contenir les données à écrire correspondant au champ pNo. La taille des données, la structure des données et leur signification peuvent dépendre du champ pNo.

4.3.2 Type de PDU long

4.3.2.1 Commande NOP et réponse

Voir 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 Commande PRM_RD et réponse

Voir 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 Commande PRM_WR et réponse

Voir 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 Commande ID_RD et réponse

Voir 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 Commande CONFIG et réponse

Voir 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 Commande ALM_RD et réponse

Voir 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 Commande ALM_CLR et réponse

Voir 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 Commande SYNC_SET et réponse

Voir 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 Commande CONNECT et réponse

Voir 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 Commande DISCONNECT et réponse

Voir 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 Commande PPRM_RD et réponse

Voir 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 Commande PPRM_WR et réponse

Voir 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 Type de PDU amélioré

4.3.3.1 Commande NOP et réponse

Voir 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 Commande PRM_RD et réponse

Voir 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 Commande PRM_WR et réponse

Voir 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 Commande ID_RD et réponse

ID_RD symbolise une commande "device-ID-read" (lecture d'ID d'appareil). À la réception de cette commande, l'esclave doit lire une partie d'un élément spécifié dans les informations relatives à l'appareil dans le champ ID_RD-RSP3-PDU.idData et doit y répondre. La sémantique et les types de données des informations relatives à l'appareil peuvent dépendre du modèle d'appareil (voir Annexe 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) }
```

a) champ idCode

idCode
Ce champ doit contenir le code d'ID d'appareil à lire.

La plage de valeurs doit être comprise entre 0 et 255. Voir B.2.2.

- '00'H: Réservé;
- '01'H: Code d'ID fournisseur;
- '02'H: Code d'appareil ou code de modèle d'appareil;
- '03'H: Version de l'appareil;
- '04'H: Version du fichier d'informations relatives à l'appareil;
- '05'H: Nombre d'adresses étendues;
- '06'H: Numéro de série du produit;
- '07'H à '0F'H: Réservé;
- '10'H: Code de profil de l'appareil pris en charge (primaire);
- '11'H: Version de profil de l'appareil pris en charge (primaire);
- '12'H: Code de profil de l'appareil pris en charge (secondaire);
- '13'H: Version de profil de l'appareil pris en charge (secondaire);
- '14'H: Code de profil de l'appareil pris en charge (tertiaire);
- '15'H: Version de profil de l'appareil pris en charge (tertiaire);
- '16'H: Cycle de transmission minimal;
- '17'H: Cycle de transmission maximal;
- '18'H: Granularité du cycle de transmission;
- '19'H: Cycle de communication minimal;
- '1A'H: Cycle de communication maximal;
- '1B'H: Nombre d'octets transmissibles;
- '1C'H: Nombre d'octets transmis (paramètre en cours);
- '1D'H: Code de profil d'appareil (paramètre en cours);
- '1E'H à '1F'H: Réservé;
- '20'H: Liste de modes de communication pris en charge;
- '21'H: Adresse MAC;
- '22'H à '2F'H: Réservé;
- '30'H: Liste de commandes principales prises en charge;
- '31'H à '37'H: Réservé;

- '38'H: Liste de sous-commandes principales prises en charge;
- '39'H à '3F'H: Réservé;
- '40'H: Liste de paramètres communs pris en charge;
- '41'H à '7F'H: Réservé;
- '80'H: Nom de l'appareil principal;
- '81'H à '8F'H: Réservé;
- '90'H: Nom du sous-appareil 1;
- '91'H à '97'H: Réservé;
- '98'H: Version du sous-appareil 1;
- '99'H à '9F'H: Réservé;
- 'A0'H: Nom du sous-appareil 2;
- 'A1'H à 'A7'H: Réservé;
- 'A8'H: Version du sous-appareil 2;
- 'A9'H à 'AF'H: Réservé;
- 'B0'H: Nom du sous-appareil 3;
- 'B1'H à 'B7'H: Réservé;
- 'B8'H: Version du sous-appareil 3;
- 'B9'H à 'BF'H: Réservé;
- 'C0'H à 'FF'H: Réservé aux informations spécifiques au fournisseur.

b) champ idOffset

idOffset
Ce champ doit contenir l'adresse de décalage de l'ID d'appareil à lire.

La plage de valeurs doit être comprise entre 0 et 255.

c) champ idSize

idSize
Ce champ doit contenir la taille de lecture en octets de l'ID d'appareil.

La plage de valeurs doit être comprise entre 0 et 255.

d) champ idData

idData
Ce champ doit contenir les données lues correspondant au champ idCode. La taille des données, la structure des données et leur signification peuvent dépendre du champ idCode. Voir B.2.2.

4.3.3.5 Commande CONFIG et réponse

Voir 4.3.1.5. Le code de "config_mode" doit être amélioré.

```
_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) }
```

a) champ config_mode

config_mode
Ce champ doit contenir le mode de configuration de l'appareil.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

– pActive ('00'H): pour procéder à un nouveau calcul avec les paramètres et la configuration;
– pAllSave ('01'H): pour écrire des paramètres dans la mémoire non volatile avec un lot;
– pReset ('02'H): pour rétablir le jeu de paramètres d'usine;
– '03'H à 'FF'H: réservé.

4.3.3.6 Commande ALM_RD et réponse

Voir 4.3.1.6. Le code du champ alarm_rd_mode et du champ alm_data correspondant doivent être améliorés.

```
_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) }
```

a) champ alm_rd_mode

alm_rd_mode
Ce champ doit contenir le mode de lecture d'alarme.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

– currentAlm ('00'H): lire l'état d'alarme/avertissement en cours;
– historyAlm ('01'H): lire l'historique d'alarme;
– cAlmDetail ('02'H): extraire les détails des informations d'alarme/avertissement individuelles en cours;
– hAlmDetail ('03'H): extraire les détails des informations individuelles de l'historique d'alarme;
– '04'H à 'FF'H: réservé.

b) champ alm_index

alm_index
Ce champ doit contenir l'index de point de lecture de l'alarme/avertissement.

La plage de valeurs doit être comprise entre 0 et 11.

c) champ alm_data

alm_data
Ce champ doit contenir l'état d'alarme lu correspondant au champ alm_rd_mode.

4.3.3.7 Commande ALM_CLR et réponse

Voir 4.3.1.7. Le code de "alm_clr_mode" doit être amélioré.

```
_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)}
```

a) champ alm_clr_mode

alm_clr_mode
Ce champ doit contenir le mode d'annulation d'alarme.

Ce champ doit être codé en type de données Unsigned8 avec la plage de valeurs suivante:

- cAlmClear ('00'H): annuler l'état d'alarme/avertissement en cours;
- hAlmClear ('01'H): annuler l'historique d'alarme;
- '02'H à 'FF'H: réservé.

4.3.3.8 Commande SYNC_SET et réponse

Voir 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 Commande CONNECT et réponse

Voir 4.3.1.9. Le champ de "profile_type" doit être amélioré.

```
_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)}
```

a) champ profile_type

profile_type
Ce champ doit contenir le code de profil d'appareil à utiliser. Comme pour le profil d'appareil, voir 4.4 et Annexe A.

La plage de valeurs doit être comprise entre 0 et 255.

4.3.3.10 Commande DISCONNECT et réponse

Voir 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 Commande PPRM_RD et réponse

Voir 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 Commande PPRM_WR et réponse

Voir 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 Commande MEM_RD et réponse

MEM_RD symbolise une commande "memory-read" (lecture de mémoire). À la réception de cette commande, l'esclave doit lire un bloc de mémoire virtuelle spécifiée dans le champ MEM_RD-RSP3-PDU.data et doit y répondre.

Comme pour la mémoire virtuelle, voir Annexe 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) }
```

a) champ mMode

mMode
Ce champ doit contenir le mode de lecture de la mémoire.

data_type
Ce champ binaire contient un type de mémoire faisant office de source pour la lecture.

'00'H: Réservé;

'01'H: Mémoire volatile;

'02'H: Mémoire non volatile;

'03'H à 'FF'H: réservé.

mode

Ce champ binaire contient le type des données spécifiées dans la mémoire à lire.

'00'H: Réservé;

'01'H: Integer8;

'02'H: Integer16;

'03'H: Integer32;

'04'H: Integer64;

'05'H à 'FF'H: réservé.

b) champ mSize

mSize
Ce champ doit contenir la quantité de données à lire dans la mémoire.

La plage de valeurs doit être comprise entre 0 et 20.

c) champ mAddress

mAddress
Ce champ doit contenir l'adresse de début de la mémoire à lire.

La plage de valeurs doit être comprise entre 0 et 'FFFF FFFF'H.

d) champ data

data
Ce champ doit contenir les données lues avec les champs mMode, mSize et mAddress spécifiés.

4.3.3.14 Commande MEM_WR et réponse

MEM_WR symbolise une commande "memory-write" (écriture de mémoire). A la réception de cette commande, l'esclave doit écrire un bloc de valeurs spécifiées de mémoire virtuelle du champ MEM_WR-CMD3-PDU.data vers la mémoire, et il doit répondre en écho pour signaler l'exécution de la commande.

Comme pour la mémoire virtuelle, voir Annexe 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)  }
```

a) champ mMode

mMode
Ce champ doit contenir le mode d'écriture de la mémoire.

data_type
Ce champ binaire doit contenir un type de mémoire de destination dans laquelle écrire.

'00'H: Réservé;

'01'H: Mémoire volatile;

'02'H: Mémoire non volatile;

'03'H à 'FF'H: réservé.

mode

Ce champ binaire doit contenir le type des données spécifiées dans la mémoire dans laquelle écrire.

'00'H: Réservé;

'01'H: Integer8;

'02'H: Integer16;

'03'H: Integer32;

'04'H: Integer64;

'05'H à 'FF'H: réservé.

b) champ mSize

mSize
Ce champ doit contenir la quantité de données de la mémoire dans laquelle écrire.

La plage de valeurs doit être comprise entre 0 et 20.

c) champ mAddress

mAddress
Ce champ doit contenir l'adresse de début de la mémoire dans laquelle écrire.

La plage de valeurs doit être comprise entre 0 et 'FFFF FFFF'H.

d) champ data

data
Ce champ doit contenir les données à écrire avec les champs mMode, mSize et mAddress spécifiés.

4.3.4 Type PDU de sous-commande

4.3.4.1 Sous-commande NOP et réponse

Voir 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 Sous-commande PRM_RD et réponse

Voir 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 Sous-commande PRM_WR et réponse

Voir 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 Sous-commande ALM_RD et réponse

Voir 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 Sous-commande PPRM_RD et réponse

Voir 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 Sous-commande PPRM_WR et réponse

Voir 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) } )
```

4.4 Profil de l'appareil

Un appareil de terrain peut fournir certains ensembles de PDU dépendantes de l'appareil de terrain et leurs CMDCodes pour le service FDC, appelés profils d'appareil de terrain. L'utilisateur d'un maître FDC peut décider lequel il convient de choisir pour la communication FDC lorsque la connexion du protocole FDC est établie.

Le système de profil d'appareil et les informations connexes sont présentés à l'Annexe A.

5 Syntaxe de transfert

5.1 Concepts

La FAL de Type 24 est une technologie d'application prioritaire. Par conséquent, il est important de faire en sorte que la taille des données codées soit compacte et de pouvoir les coder et les décoder simplement.

Les processus de codage et de décodage doivent donc être définis de manière aussi spécifique à la FAL de Type 24, cela n'impliquant aucun caractère commun avec l'autre type de protocole ni aucune versatilité. Le format de données et la règle de codage de l'APDU doivent être préalablement définis pour les entités qui transfèrent les données, cela n'impliquant aucun service de couche de présentation.

Dans la règle de codage de la FAL de Type 24, on peut ne pas coder la balise (code de type de données) et la longueur de données. Les seules valeurs de chaque type de données défini avec la syntaxe abstraite ASN.1 doivent être codées dans une ligne de données.

La longueur de la PDU doit être fixe dans le cas d'un service FDC dont le facteur temps est particulièrement important. Pour un service MSG dont le facteur temps n'est pas strict, la PDU de longueur variable peut être transférée. Dans ce cas, il convient que la syntaxe abstraite définisse explicitement un champ de données précisant la longueur de données.

Il convient que la couche inférieure définisse l'ordre des bits sur la voie de transmission.

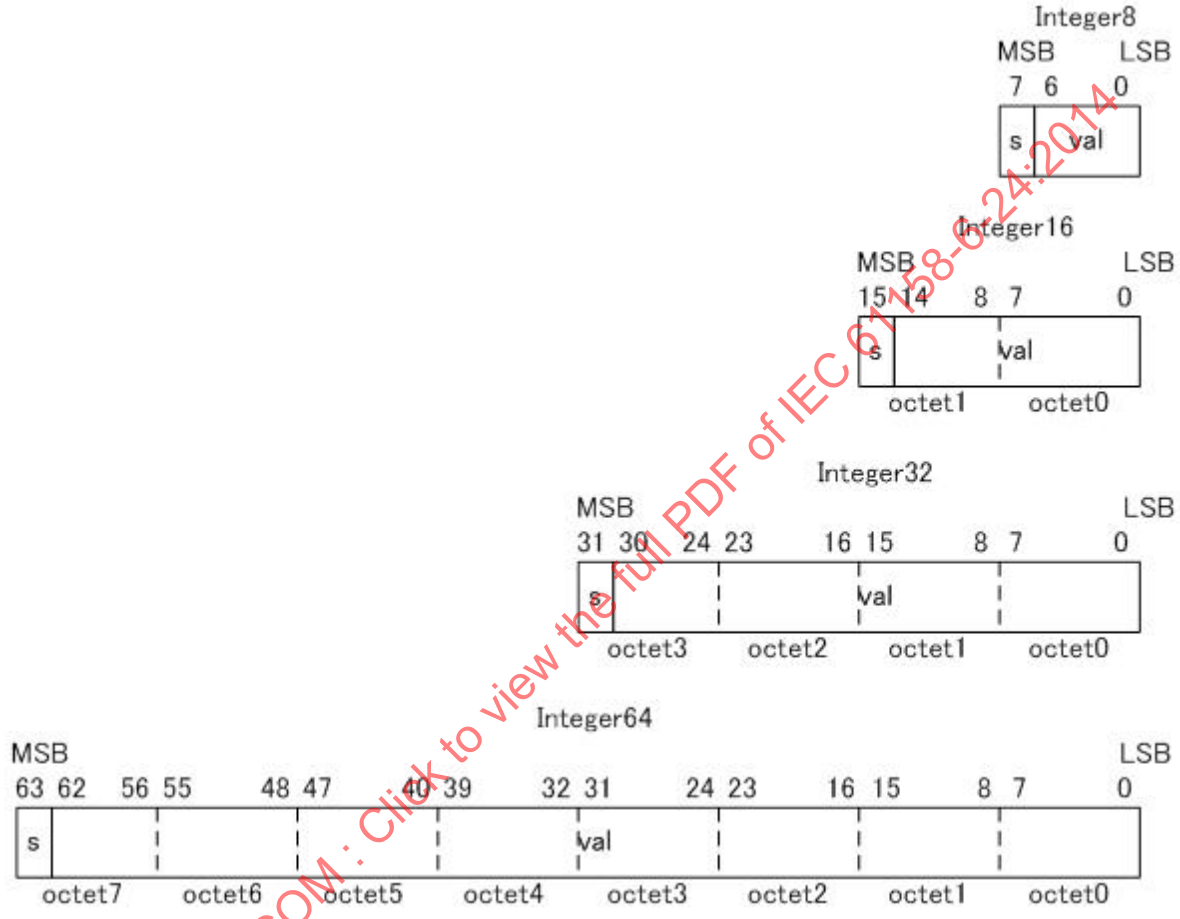
5.2 Règles de codage

5.2.1 INTEGER et ses sous-types

INTEGER et ses dérivés (Integer8, Integer16, Integer32, Integer64, Unsigned8, Unsigned16, Unsigned32 et Unsigned64) doivent coder et transférer uniquement la valeur de données dans leur taille d'octet.

La forme codée d'Integer8, Integer16, Integer32 et Integer64 doit être composée d'un bit:*s* de signe sur le bit de poids fort (voir Figure 2). Le *val* doit contenir la valeur elle-même s'il est supérieur ou égal à 0 (le bit de signe est '0'B). Si *val* est un nombre négatif (le bit de signe est '1'B), les données codées doivent être représentées comme un complément de 2 à l'aide de *s* et *val*.

NOTE Cette règle de codage s'applique à la structure de données lors du transfert de la PDU, mais on peut ne pas préciser la structure de données sur une mémoire réelle du système de traitement des données d'un appareil réel.



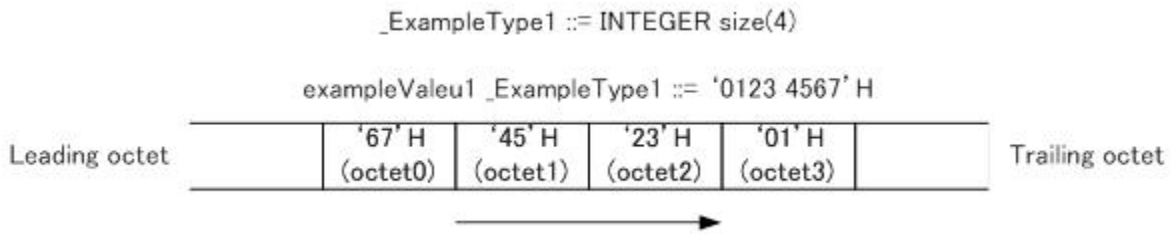
s: bit de signe (0: + / 1: -)

val: valeur positive, si *s*=0;

"*s*+*val*" représente une valeur négative en complément de 2, si *s*=1.

Figure 2 – Codage des sous-types Integer

Lors du transfert des données d'un type de données à plusieurs octets, les octets doivent être transmis dans l'ordre, du plus bas vers le plus élevé (voir l'exemple à la Figure 3).



Légende

Anglais	Français
Leading octet	Octet de départ
Trailing octet	Octet de fin

Figure 3 – Exemple de transfert de la valeur INTEGER

Au format codé Unsigned8, Unsigned16, Unsigned32 et Unsigned64, les données de valeur numérique (*val*) doivent être définies dans la zone de données de sa taille d'octet (voir Figure 4). Dans ce cas, la valeur numérique doit être définie de manière à placer le numéro d'un chiffre inférieur dans un bit inférieur de la séquence, la valeur 0 devant être définie sur le bit supérieur non utilisé.

Lors du transfert des données d'un type de données à plusieurs octets, les octets doivent être transmis dans l'ordre, du plus bas vers le plus élevé.

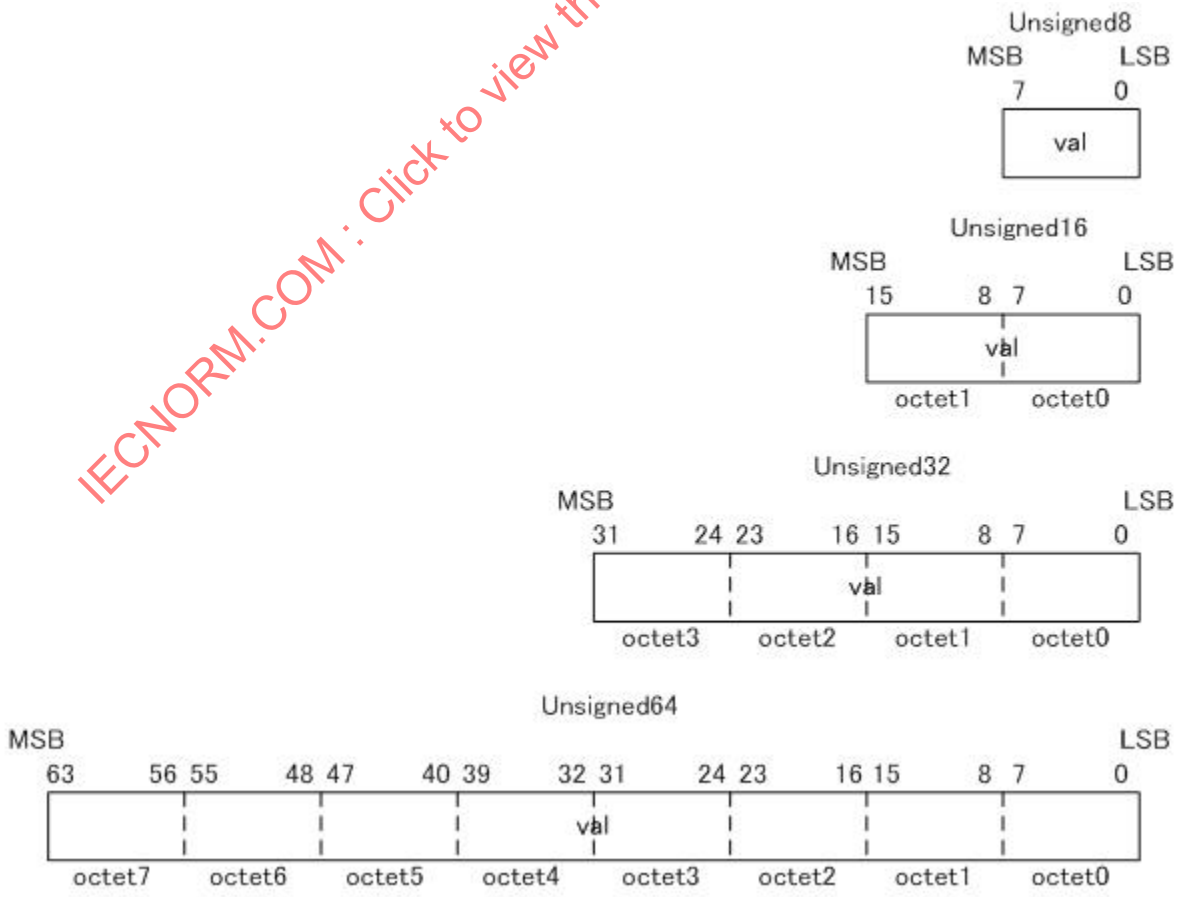


Figure 4 – Codage des sous-types Unsigned

5.2.2 Type REAL et ses sous-types

La FAL de Type 24 ne définit pas directement les données du type REAL dans la syntaxe abstraite et utilise Float32 et Float64, qui sont dérivées du type REAL. Ce codage doit satisfaire à l'ISO/CEI/IEEE 60559. Leurs formats codés sont présentés à la Figure 5 et à la Figure 6.

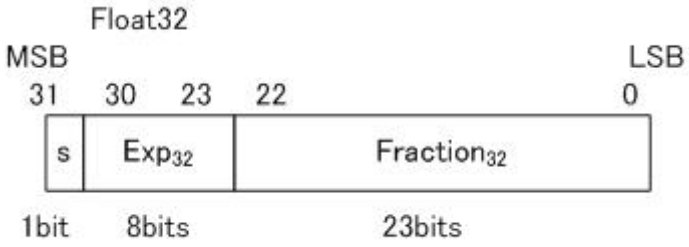


Figure 5 – Codage du type Float32

Les données du type Float32 doivent être composées de 4 octets (voir Figure 4). Les octets doivent être transmis dans l'ordre, du plus bas au plus élevé. Dans ce cas, la valeur des données à virgule flottante du type Float32 doit être calculée à l'aide de la formule suivante:

$\text{float32} = (-1)^s \times c_{32} \times 2^{q_{32}}$,

où

s: bit de signe (0:+ / 1:-),

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

$= 1 + \frac{b_{22}}{2} + \frac{b_{21}}{2^2} + \cdots + \frac{b_0}{2^{23}}$: mantisse,

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

Si $Fraction_{32}$ et Exp_{32} sont égaux à 0, $float_{32}$ représente 0.

Si $Fraction_{32}$ est égal à 0 et que Exp_{32} est égal à 255, $float_{32}$ représente l'infinie positive ou négative comme le bit de signe s.

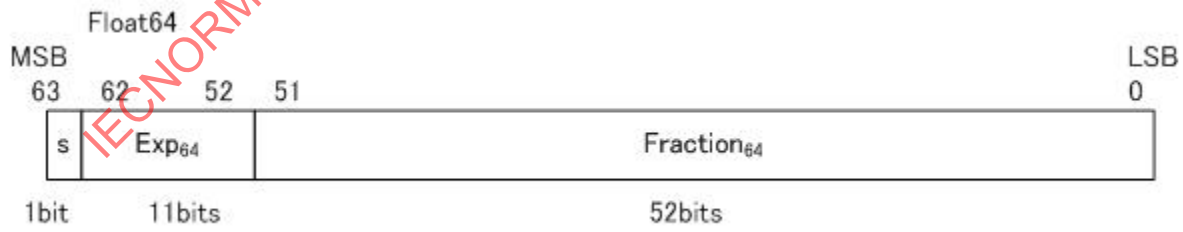


Figure 6 – Codage du type Float64

Les données du type Float64 doivent être composées de 8 octets (voir Figure 6). Les octets doivent être transmis dans l'ordre, du plus bas au plus élevé. Dans ce cas, la valeur des données à virgule flottante du type Float64 doit être calculée à l'aide de la formule suivante:

$\text{float64} = (-1)^s \times c_{64} \times 2^{q_{64}}$,

où

s: bit de signe (0:plus / 1:moins),

$c_{64} = \left(1 \times 2^{52} + Fraction_{64}\right) \times 2^{-52}$

$= 1 + \frac{b_{51}}{2} + \frac{b_{50}}{2^2} + \cdots + \frac{b_0}{2^{52}}$: mantisse,

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

Si $Fraction_{64}$ et $Exp64$ sont égaux à 0, $float_{64}$ représente 0.

Si $Fraction_{64}$ est égal à 0 et que Exp_{64} est égal à 2 047, $float_{64}$ représente l'infinie positive ou négative comme le bit de signe s.

5.2.3 Type BIT STRING

Dans le type BIT STRING, le bit de début (0ème bit) doit correspondre au bit de poids faible des données codées, le bit de fin devant correspondre au bit de poids fort en fonction des nombres associés aux bits nommés. Toutefois, les données de remplissage doivent être remplies par une frontière d'octet. Dans ce cas, une zone peut être réservée même pour le bit non défini. Toutefois, la valeur du bit n'est pas définie. La Figure 7 présente un exemple de définition de champ binaire du type BIT STRING.

Pour un bloc de données BIT STRING de taille de données à plusieurs octets, les octets doivent être transmis dans l'ordre, à partir de l'octet contenant le bit le plus bas.

NOTE La définition de "bit de début" et "bit de fin" est conforme à l'ISO/CEI 8824-1:2008, 22.2.

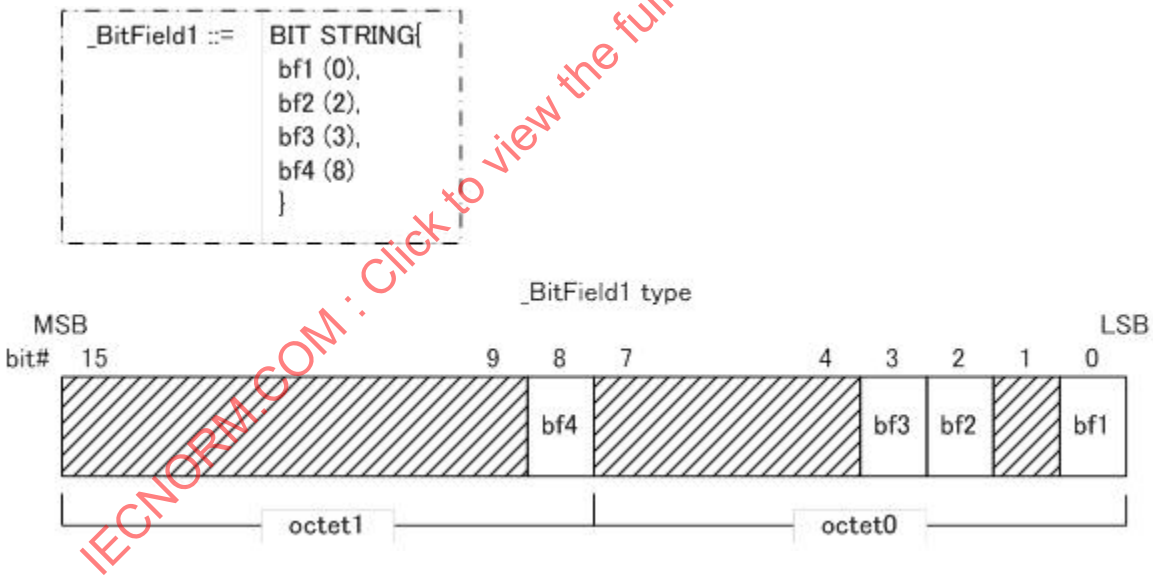


Figure 7 – Exemple de définition de champ binaire avec bits nommés

Un champ binaire peut être défini en combinant les types SEQUENCE et BIT STRING. Dans ce cas, la frontière de chaque champ doit être définie en fonction de la taille de bit spécifiée et remplie à partir du bit le plus faible (voir l'exemple présenté à la Figure 8).

Pour coder ce champ binaire, un octet doit être rempli à partir de son bit le plus faible, en fonction de l'ordre prévu par la définition de champ décrite entre "{" et "}" de la syntaxe SEQUENCE afin de définir une zone de données confinée par la frontière d'octet. S'il reste des bits de fraction non définis, il doit s'agir d'une zone réservée et leurs valeurs doivent être indéfinies.

```
_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)
                }
```

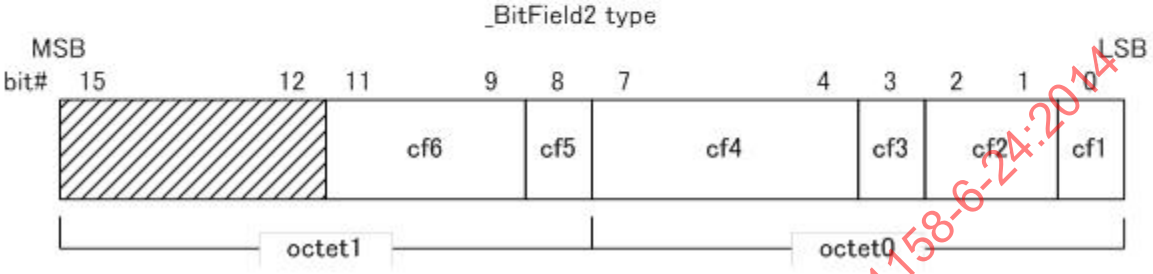


Figure 8 – Exemple de définition de champ binaire avec taille de champ

5.2.4 Type OCTET STRING et type IA5String

Les données de type OCTET STRING doivent être transférées sans convertir le code dans l'ordre à partir de l'octet de poids fort des données de chaîne données.

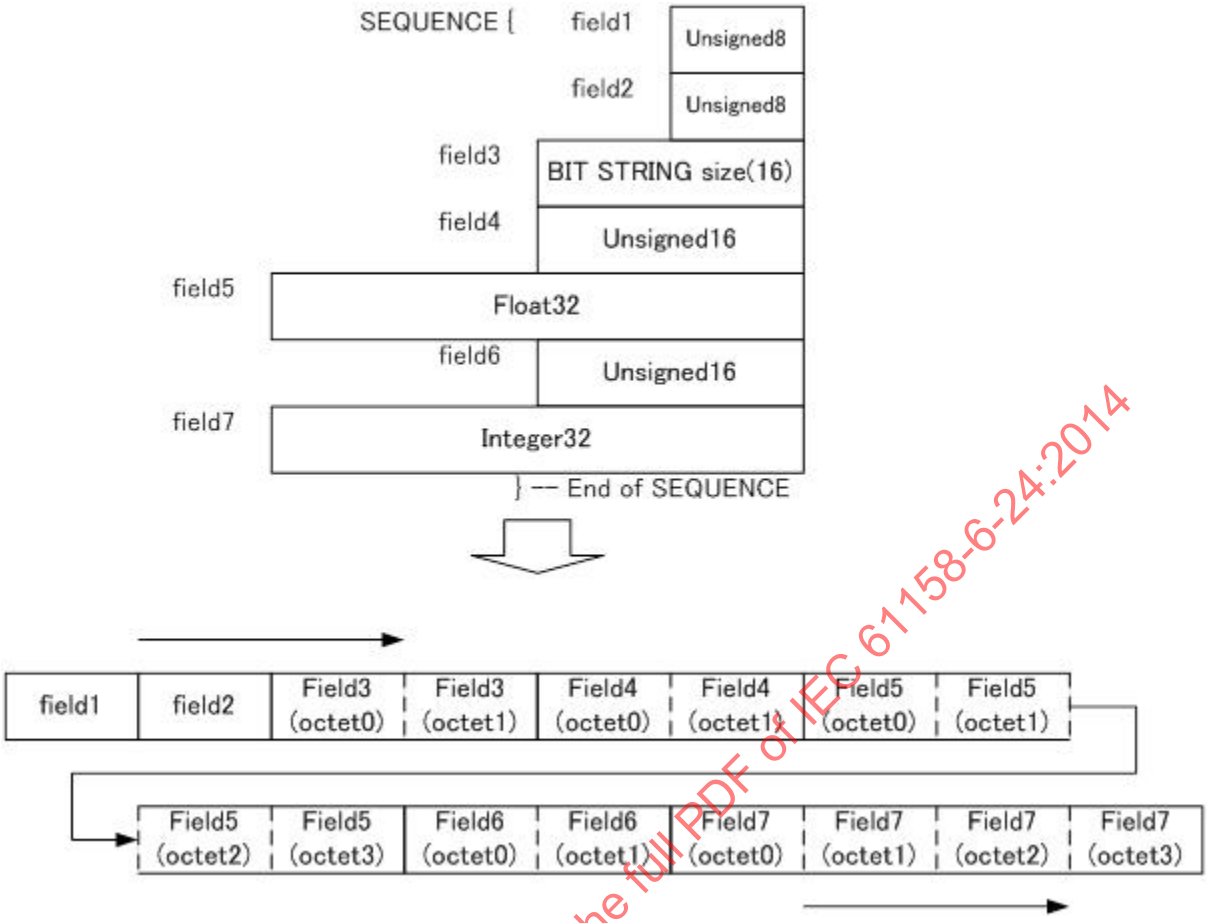
Les données de type IA5String doivent être codées dans des lignes de données (OCTET STRING) en convertissant les données de chaîne en un code à 1 octet, conformément à l'ISO/CEI 646, caractère par caractère, puis en ajoutant un octet de code nul ('00'H) au dernier code de caractère. Les données doivent être transférées dans l'ordre à partir des données de code du premier caractère.

5.2.5 Type NULL

Pour les données de type NULL, la longueur de données est 0 et aucune valeur n'existe. Par conséquent, elles ne doivent pas être codées ni transférées.

5.2.6 Type Structure et type Array

Parmi les éléments du type Structure, le type SEQUENCE et le type SEQUENCE OF ne doivent pas être codés. Les composants inclus dans ces types doivent être codés et transférés conformément à chaque règle et à l'ordre de transfert d'octet (voir Figure 9).



Légende

Anglais	Français
End of SEQUENCE	Fin de SEQUENCE

Figure 9 – Codage du type SEQUENCE

De même, le type CHOICE ne doit pas être codé. Pour modifier les données de communication dans l'une des alternatives de ce type, il convient de sélectionner les données en fonction du contexte de l'AP de communication et il convient d'établir un accord entre les appareils de communication. La méthode peut dépendre de la mise en œuvre et n'entre pas dans le domaine d'application de la présente Norme. Les données doivent être codées en fonction de la règle du composant sélectionné.

6 Structure du diagramme d'états de protocole FAL

Dans le présent article et ceux qui suivent, les FAL ASE sont caractérisés avec des modèles de diagramme d'états de protocole (PM).

Bien que ce type de bus de terrain dispose de sa structure de PM, une table de mapping est présentée ci-dessous pour la clarté de la compréhension (voir Tableau 4). Elle mappe la structure de ce type à la structure à quatre couches classique adoptée par la plupart des autres types, contenant la PM de contexte AP, la FSPM, l'ARPM et la DMPM.

La Figure 10 présente la structure des PM.

– Il existe un diagramme d'états de contexte AP défini de manière formelle (APC SM) permettant à l'utilisateur FAL de lancer une application de communication particulière.

– Il n'existe aucune définition de FSPM faisant simplement office d'interface entre l'utilisateur FAL et l'ARPM. Seuls deux types de machines protocolaires sont définis:
  - un ensemble de machines de commande d'appareil de terrain (FDC PM) faisant office de maître, d'esclave et de moniteur pour les utilisateurs du service FDC;
  - une paire de machines de message (MSG PM) faisant office de demandeur et de répondeur pour les utilisateurs de service de message.
– Deux types différents de machine ARPM sont définis au niveau de l'interface de la Couche liaison de données (DLL):
  - un ensemble de machines ARPM pour les relations entre applications orientée connexion entre les classes de maître, d'esclave ou de moniteur de service FDC;
  - un type de machine ARPM pour les relations entre applications sans connexion entre les classes de demandeur et de répondeur du service de messages.
– Il n'existe aucune définition formelle de la machine protocolaire de mapping DLL (DMPM), unifiée dans l'ARPM ou l'APCSM. Les services DL sont utilisés directement par l'ARPM ou l'APCSM.

La FDCPM est censée fonctionner dans le FDC ASE. De même, la MSGPM est censée fonctionner dans le MSG ASE, l'ARPM dans l'AR ASE et l'AP-CONTEXT SM dans le FSM ASE.

Tableau 4 – Mapping des diagrammes d'états de protocole

Type d'AP	ASE	Classe activée	Machine protocolaire	Mapping à une structure classique pour la plupart des autres types
AP du maître C1	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 avec DMPM
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a
AP de l'esclave	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 avec DMPM
		FDCMonitor-AR	ARPM-FDCMN	
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a
AP du maître C2	FSM ASE	FieldbusSystemManager	APC SM	AP-Context-PM
	FDC ASE	Monitor	FDC PM-MN	FSPM
	MSG ASE	Requester	MSG PM-RQ	FSPM
		Responder	MSG PM-RS	
	AR ASE	FDCMonitor-AR	ARPM-FDCMN	ARPM avec DMPM
		Message-AR	ARPM-MSG	
	EVM ASE	EventManager	n/a	n/a

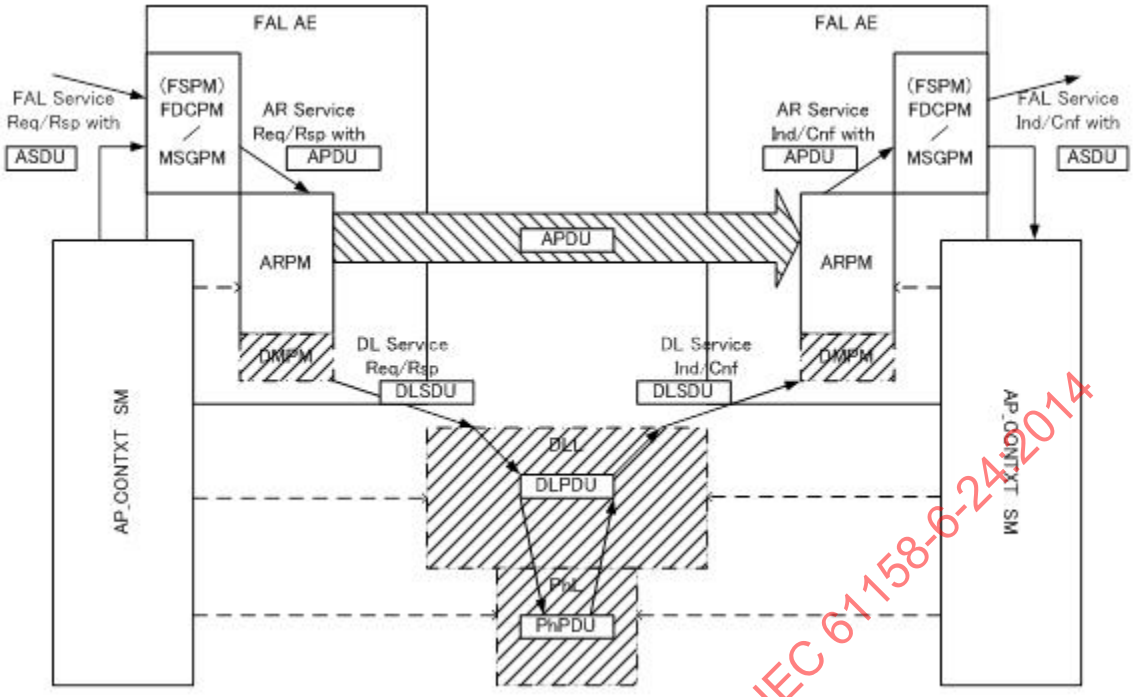


Figure 10 – Structure des diagrammes d'états de protocole FAL

7 Diagramme d'états de contexte AP (APC SM)

7.1 Présentation

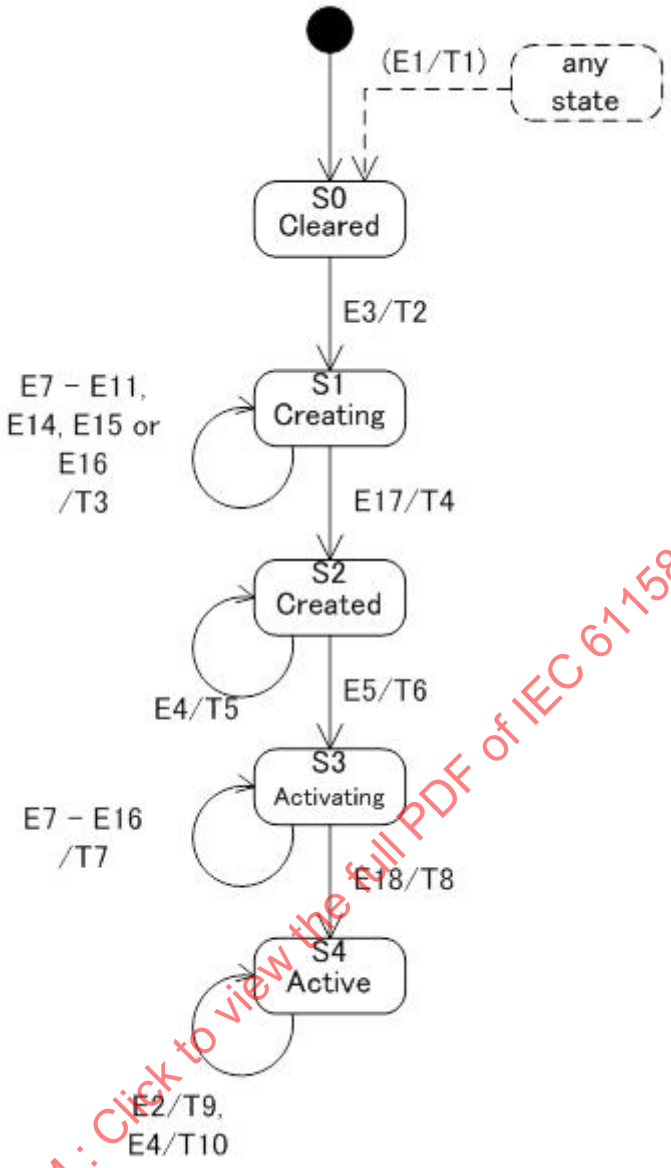
Un contexte d'AP est un ensemble d'informations et de règles relatives à un système de communication. Il doit être créé par des instructions provenant d'un utilisateur FAL à l'appel du processus d'application (AP). Par exemple, il peut contenir les paramètres de communication sélectionnés, en fonction de la configuration du système, un domaine d'application du système, le profil de l'appareil et les différences d'aptitudes de communication entre l'appareil et ses homologues.

Dans ce modèle ASE, il convient qu'une FAL conserve les informations mentionnées ci-dessus dans les attributs de la classe FieldbusSystemManager, conformément aux instructions et configurations de l'utilisateur. Il convient que les informations puissent permettre de caractériser les types d'AP (le maître C1, le maître C2, l'esclave et l'esclave de surveillance par exemple). De plus, il convient de pouvoir configurer d'autres paramètres de communication et le profil d'appareil (un servo-commande, un onduleur ou un appareil d'E/S, par exemple) afin d'établir l'environnement de communication.

L'APCSM doit contrôler une série de transitions d'état dans lesquelles un appareil démarre. L'APCSM doit obtenir les données de contexte AP auprès de l'utilisateur FAL et doit initialiser la FAL et la couche inférieure avec le contexte. En conséquence, elle doit établir une communication constante. L'APCSM réalise le service que le FSM ASE offre.

L'APCSM peut ne pas modifier directement une APDU ni ne réalise un protocole pour transférer des PDU. Toutefois, elle peut indirectement agir comme un déclencheur d'échange de PDU en utilisant l'autre service ASE et le service de couche inférieure lors du processus d'initialisation de la FAL.

La Figure 11 présente le schéma d'états APCSM.



Légende

Anglais	Français
any state	n'importe quel état
Cleared	Annulé
Creating	En cours de création
Created	Créé
Activating	En cours d'activation
Active	Actif
E7 – E11, E14, E15 or E16 / T3	E7 – E11, E14, E15 ou E16 / T3

Figure 11 – Schéma d'états de l'APCSM

7.2 Descriptions des états

Le Tableau 5 décrit chaque état de l'APCSM.

Tableau 5 – Descriptions des états de l'APC SM

S#	État	Sous-état	Description
S0	Annulé	-	État qui est celui de la classe FieldbusSystemManager du FSM ASE qui vient d'être instancié comme ce diagramme d'états Toutes les entités de chaque couche et les ASE de l'appareil sont réinitialisés, et le contexte AP est annulé.
S1	En cours de création		État transitoire dans lequel le contexte AP ou l'environnement de communication est généré, selon l'entrée CONTEXT-DATA et le résultat de la négociation entre les appareils correspondants
S2	Créé	-	État dans lequel le contexte AP a été généré
S3	En cours d'activation	-	État transitoire dans lequel les entités de chaque couche et les ASE sont activés de manière séquentielle
S4	Actif	-	État permanent dans lequel toutes les entités sont activées et les services normaux de communication fournis

7.3 Événements déclencheurs

Le Tableau 6 répertorie chaque événement déclencheur de l'APCSM.

Tableau 6 – Événements déclencheurs de l'APC SM

E#	Événement déclencheur: primitive ou condition	Source de l'événement	Paramètres associés	Description
E1	FSM-Reset.req	Utilisateur FAL		
E2	FSM-GetStatus.req	Utilisateur FAL	INFO-ID	
E3	FSM-SetContext.req	Utilisateur FAL	CONTEXT-DATA	
E4	FSM-GetContext.req	Utilisateur FAL		
E5	FSM-Start.req	Utilisateur FAL		
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 (état S1)		Événement interne
E18	APC-Activated	APCSM (état S3)		Événement interne

7.4 Descriptions des actions aux transitions d'état

Les spécifications particulières selon la mise en œuvre n'entrent pas dans le domaine d'application de la présente Norme (le processus d'initialisation, la temporisation d'activation et les différents paramètres de communication de chaque couche et des ASE, par exemple). L'action connexe est simplement présentée au Tableau 7.

Tableau 7 – Transitions de l'APC SM

T#	État source	Événement (arguments) [conditions] / action	État cible
T1	(n'importe quel état)	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:Annulé
T2	S0: Annulé	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: En cours de création
T3	S1: En cours de création	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: En cours de création
T4	S1: En cours de création	E17:APC-Created / FSM-SetContext.cnf;	S2: Créé
T5	S2: Créé	E4:FSM-GetContext.req / FSM-GetContext.cnf;	S2: Créé
T6	S2: Créé	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: En cours d'activation
T7	S3: En cours d'activation	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: En cours d'activation
T8	S3: En cours d'activation	E18: APC-Activated / FSM-Start.cnf;	S4: Actif
T9	S4: Actif	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: Actif
T10	S4: Actif	FSM-GetContext.req / FSM-GetContext.cnf;	S4: Actif

8 Machines protocolaires de service FAL (FSPM)

8.1 Présentation

Si la FSPM reçoit une primitive de demande de service ou une primitive de réponse provenant d'un utilisateur FAL, elle doit modifier une APDU provenant d'une SDU comme le paramètre des primitives, puis demander à l'ARPM de la transmettre. Elle doit également retirer une SDU de l'APDU reçue par l'ARPM afin de la livrer à l'utilisateur FAL comme une primitive d'indication ou une primitive de conformité.

Deux types de services d'application (FDC ASE et MSG ASE) peuvent être proposés à l'utilisateur FAL dans la FAL de Type 24 (voir CEI 61158-5-24), et la FSPM peut être composée de deux types de PM (FDC PM et MSG PM) correspondant aux ASE.

8.2 Machine protocolaire de commande d'appareil de terrain (FDC PM)

8.2.1 Présentation du protocole

La FDC PM est un diagramme d'états de protocole (PM) qui réalise les services fournis par le FDC ASE. Les protocoles sont classés comme indiqué au Tableau 8.

Tableau 8 – Mode de protocole FDC

Mode de transmission (par la DLL)	État de communication	Type de commande communication	Description
Cyclique	Communication synchrone (Sync)	Commande de type de communication synchrone (commande synchrone)	Le FDC ASE (maître et esclave) doit informer l'utilisateur du déclenchement d'un événement dans chaque cycle de communication avec FDC-ComCycle.ind. L'utilisateur du maître FDC peut mettre à jour une commande suivante vers l'AP esclave dans chaque cycle de communication pour demander son transfert sans attendre la réponse de fin de processus (FDC-Command.cnf) adressée à la commande précédente. L'esclave FDC doit recevoir une commande de la part du maître dans chaque cycle de communication et la transmettre à l'utilisateur de l'esclave FDC avec FDC-Command.ind. Il convient que l'utilisateur traite la commande, puis renvoie la réponse (FDC-Command.rsp) dans le cadre d'un cycle de communication.
		Commande de type de communication asynchrone (commande asynchrone)	Le FDC ASE (maître et esclave) doit informer l'utilisateur du déclenchement d'un événement dans chaque cycle de communication avec FDC-ComCycle.ind. Il convient que l'utilisateur du maître FDC demande le transfert d'une commande vers l'AP esclave dans chaque cycle de communication en attendant la réponse de fin de processus adressée à la commande précédente. Il convient de conserver le contenu de la commande tant que l'utilisateur n'a pas reçu la réponse de fin de processus. L'esclave FDC doit recevoir une commande de la part de l'AP maître dans chaque cycle de communication et la transmettre à l'utilisateur. Il convient que l'utilisateur renvoie une réponse correspondante (FDC-Command.rsp) avec un état de progression de la commande (cmdRdy) dans chaque cycle de communication, et il convient qu'il n'accepte pas une nouvelle commande tant que la commande de traitement n'est pas terminée.

Mode de transmission (par la DLL)	État de communication	Type de commande communication	Description
	Communication asynchrone (Async)	Commande asynchrone	Le FDC ASE (maître et esclave) doit informer l'utilisateur du déclenchement d'un événement dans chaque cycle de communication avec FDC-ComCycle.ind. Lorsque l'utilisateur du maître FDC demande le transfert d'une commande vers l'AP esclave, il peut attendre la réponse de fin de processus correspondante adressée à la commande précédente. En attendant, il convient que l'utilisateur ne demande pas le transfert d'une commande mise à jour. L'esclave FDC peut recevoir une commande provenant de l'AP maître dans chaque cycle de communication. L'esclave FDC doit transmettre la commande à l'utilisateur suite à la mise à jour de son contenu. Il convient que l'utilisateur renvoie une réponse correspondante (FDC-Command.rsp) avec un état de progression de la commande (cmdRdy) dans chaque cycle de communication. Il ne doit pas accepter une nouvelle commande tant que la commande de traitement n'est pas terminée.
Event-driven (déclenché par l'événement)	Async	Commande asynchrone	Le FDC ASE (maître et esclave) ne doit pas informer l'utilisateur d'un événement dans un cycle normal, car aucun cycle de communication n'est généré. Si l'utilisateur du maître FDC demande le transfert d'une commande, il convient qu'il attende la réponse de fin de processus correspondante pour la commande précédente et il convient qu'il ne transfère pas de commande mise à jour. L'esclave FDC doit transmettre à l'utilisateur la commande reçue de l'AP maître. Il convient que l'utilisateur ne renvoie pas de réponse ni n'accepte de nouvelle commande tant que le traitement de la commande reçue n'est pas terminé ou que la durée de traitement prédéfinie n'est pas écoulée.

8.2.2 Mode de communication cyclique

8.2.2.1 Spécifications communes de communication cyclique

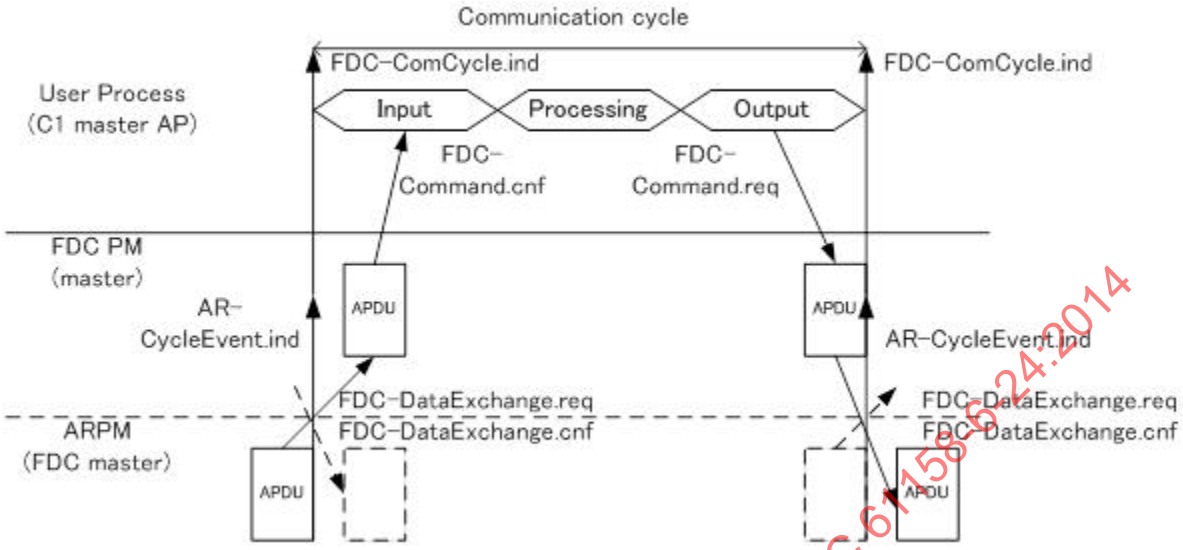
En mode de communication cyclique, un événement doit être généré dans une période constante, appelée cycle de communication. Le processus de communication fourni par le FDC ASE doit être exécuté de manière répétée avec cet événement. Le cycle de communication doit émaner du cycle d'un multiple entier du cycle de transmission géré par la Couche liaison de données.

Il convient que le cycle de transmission soit un événement de cycle constant généré avec le compteur d'exécution périodique synchronisé avec tous les appareils.

Le cycle de communication doit commencer une fois la connexion établie entre le maître et l'esclave. Par conséquent, avant d'établir la connexion (même en mode de communication cyclique), le processus de communication ne peut pas être synchronisé avec le cycle de communication (sauf le cycle primaire ou le cycle de transmission), seule la communication s'appuyant sur certaines commandes de type asynchrone pouvant être exécutée.

Le protocole d'établissement et de libération de la connexion doit également être fourni sous la forme de services FDC ASE.

Le cycle de communication du FDC ASE peut avoir un impact sur le processus utilisateur (l'AP maître C1 ou l'AP esclave, par exemple). Voir Figure 12, Figure 13.



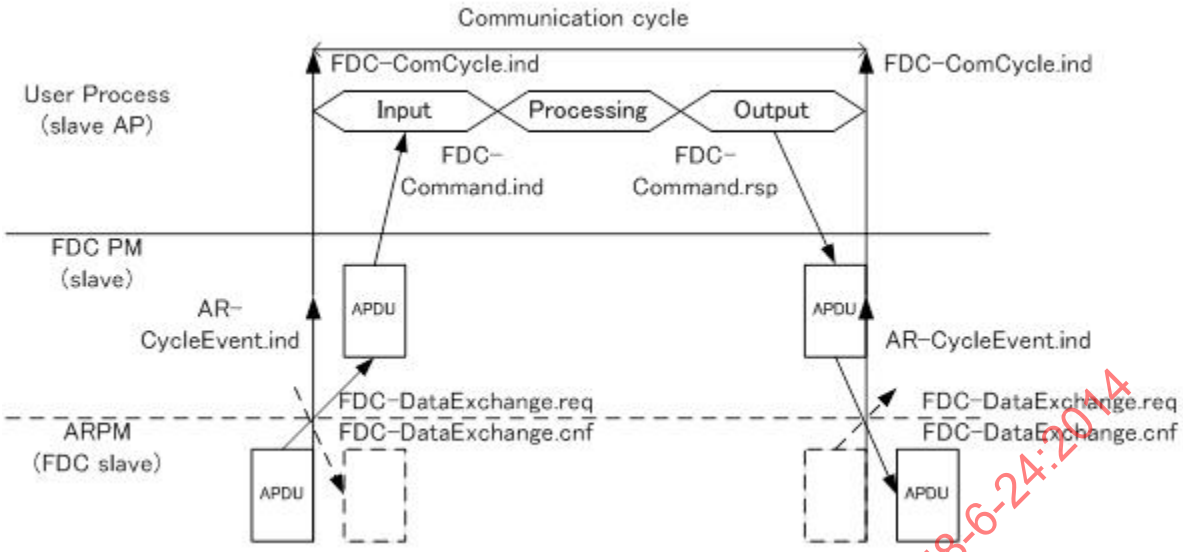
Légende

Anglais	Français
Communication cycle	Cycle de communication
User Process (C1 master AP)	Processus utilisateur (AP maître C1)
Input	Entrée
Processing	Traitement
Output	Sortie
FDC PM (master)	FDC PM (maître)
ARPM (FDC master)	ARPM (maître FDC)

Figure 12 – Exemple de cycle de communication de l'AP maître FDC

Par exemple, dans le maître C1, la synchronisation de sortie de commande, la synchronisation d'entrée de réponse et la synchronisation de traitement des données peuvent être gérées de manière plus efficace si elles sont traitées avec l'événement de cycle (FDC-ComCycle.ind), comme indiqué à la Figure 12.

NOTE La Figure 11 et la Figure 12 sont présentées à titre informatif.



Légende

Anglais	Français
Communication cycle	Cycle de communication
User Process (slave AP)	Processus utilisateur (esclave AP)
Input	Entrée
Processing	Traitement
Output	Sortie
FDC PM (slave)	FDC PM (esclave)
ARPM (FDC slave)	ARPM (esclave FDC)

Figure 13 – Exemple de cycle de communication de l'AP esclave FDC

De la même manière pour l'AP esclave, la temporisation d'entrée de commande, la temporisation de sortie de réponse et la temporisation de traitement des données peuvent être gérées de manière plus efficace si elles sont traitées comme indiqué à la Figure 13.

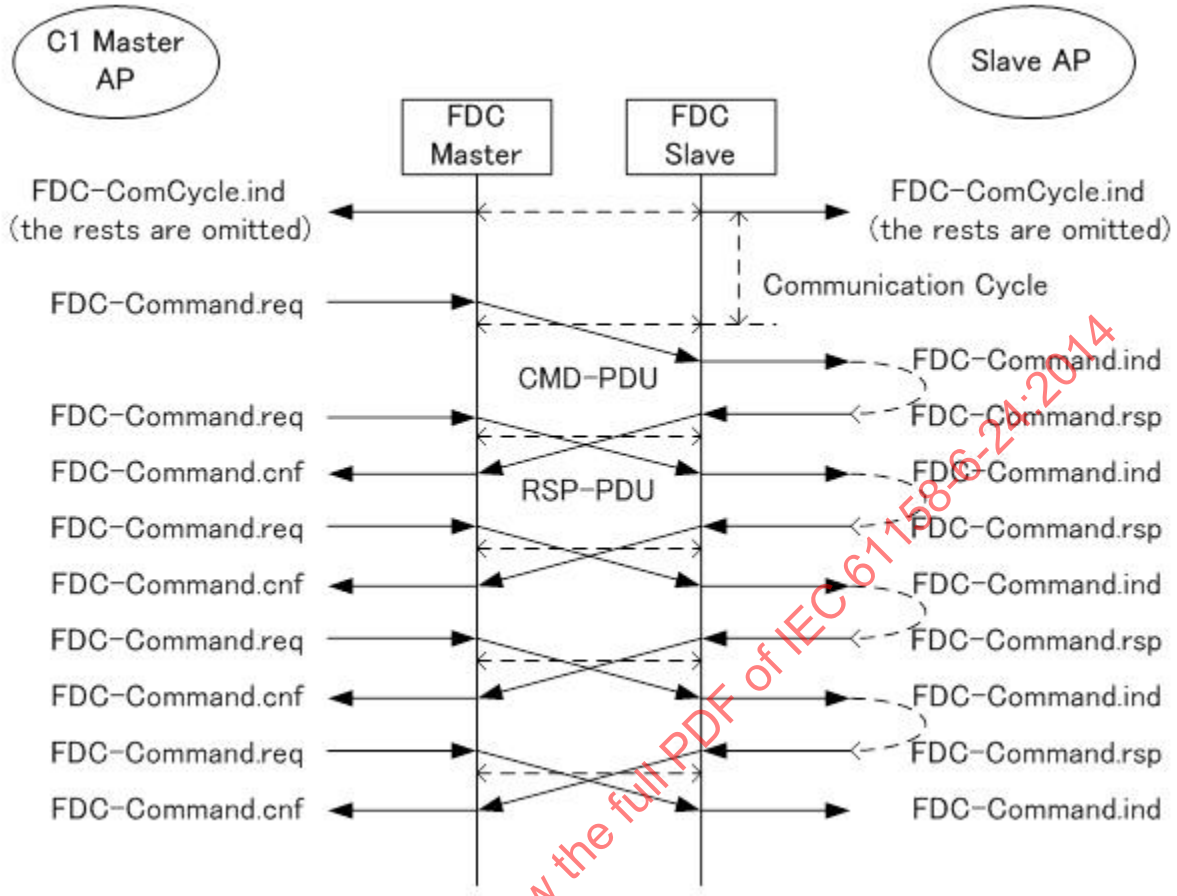
8.2.2.2 État de communication synchrone (SyncConnected)

Le mot "synchrone" utilisé dans les expressions "état de communication synchrone" et "commande de type de communication synchrone" signifie qu'il convient de lancer ou d'exécuter de manière cyclique les processus utilisateur qui émettent les commandes dans ces états, et de synchroniser le processus de communication FDC ASE avec le cycle de communication avant de l'exécuter périodiquement.

Il convient que l'AP maître et l'AP esclave demandent respectivement le transfert d'une commande et le transfert d'une réponse une seule fois dans chaque cycle de communication lorsque la FDCPM est à l'état de communication synchrone. Dans ce cas, il convient que l'AP maître et l'AP esclave surveillent mutuellement l'état de leur activité de synchronisation grâce à un compteur ou un champ _WDT (horloge de surveillance) sur chaque SDU.

Si l'AP maître transfère au moins deux fois sans interruption des commandes de type synchrone, il peut demander à les transférer l'une après l'autre dans chaque cycle de communication sans attendre de recevoir la réponse correspondante. Il convient que l'AP esclave traite la commande synchrone dans le cadre du cycle de communication dans lequel elle a été reçue, puis transmette la réponse. L'AP maître et l'AP esclave peuvent échanger ce type de commande et de réponse avec le compteur WDT afin de reconnaître qu'ils sont synchronisés l'un avec l'autre.

La Figure 14 présente le chronogramme de la communication de commande synchrone.



Légende

Anglais	Français
C1 Master AP	AP maître C1
FDC Master	Maître FDC
FDC Slave	Esclave FDC
Slave AP	AP esclave
(the rests are omitted)	(le reste est ignoré)
Communication Cycle	Cycle de communication

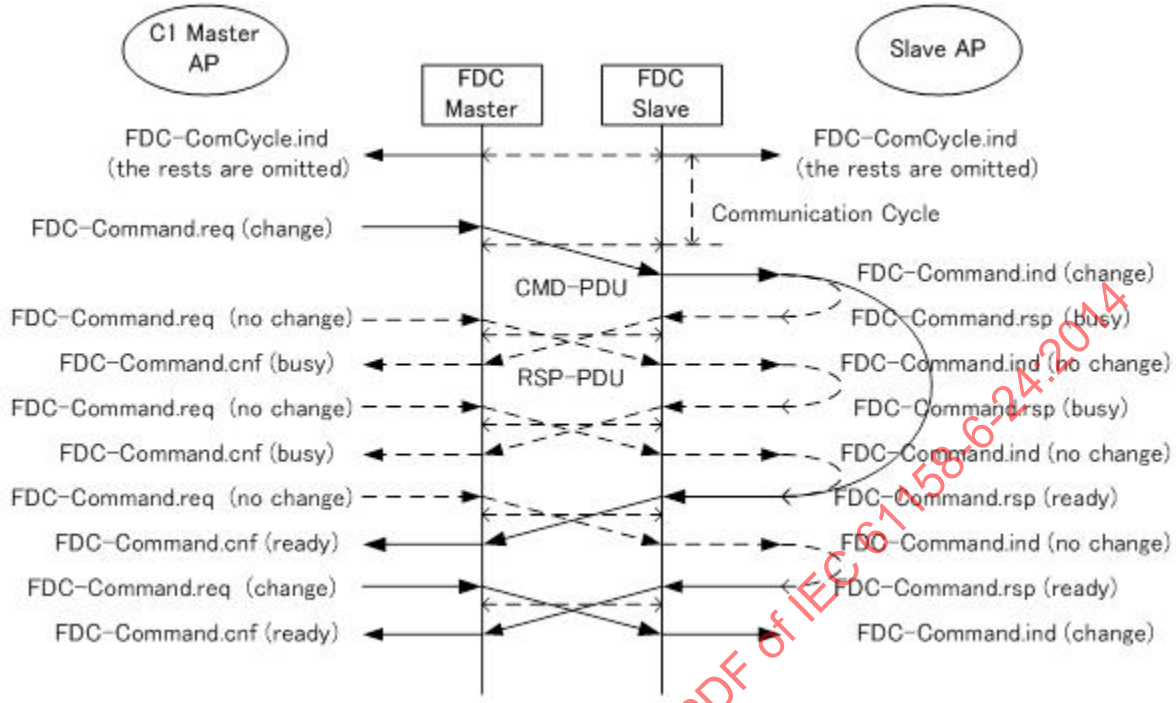
Figure 14 – Communication de commande synchrone à l'état de synchronisation

Si l'AP maître transfère une commande de type asynchrone, il convient qu'il demande le transfert de la commande suivante après avoir confirmé la réponse correspondante à la commande précédente, en indiquant que le processus s'est terminé par "cmdRdy-bit = 1" ou "commande prête".

Toutefois, même dans ce cas, il convient que l'AP maître demande en permanence le transfert de la commande avec le même contenu dans chaque cycle de communication. Il convient que l'AP esclave demande le transfert de la réponse avec un "cmdRdy-bit = 0" ou une "commande occupée", de sorte que chaque surveillance de l'état synchronisé grâce au WDT soit un succès.

NOTE Les termes tels que "commande de type synchrone" et "commande de type asynchrone" permettent de préciser si le processus utilisateur est exécuté lors de la synchronisation avec le cycle de communication ou pas. D'autre part, du point de vue de la gestion des transactions pour la commande et la réponse, on peut affirmer que la commande de type synchrone traite les transactions asynchrones, et inversement.

La Figure 15 présente le chronogramme de la communication de commande asynchrone à l'état synchrone.



Légende

Anglais	Français
C1 Master AP	AP maître C1
FDC Master	Maître FDC
FDC Slave	Esclave FDC
Slave AP	AP esclave
(the rests are omitted)	(le reste est ignoré)
Communication Cycle	Cycle de communication
(no change)	Aucun changement
(busy)	Occupé
(ready)	Prêt

Figure 15 – Communication de commande asynchrone à l'état sync

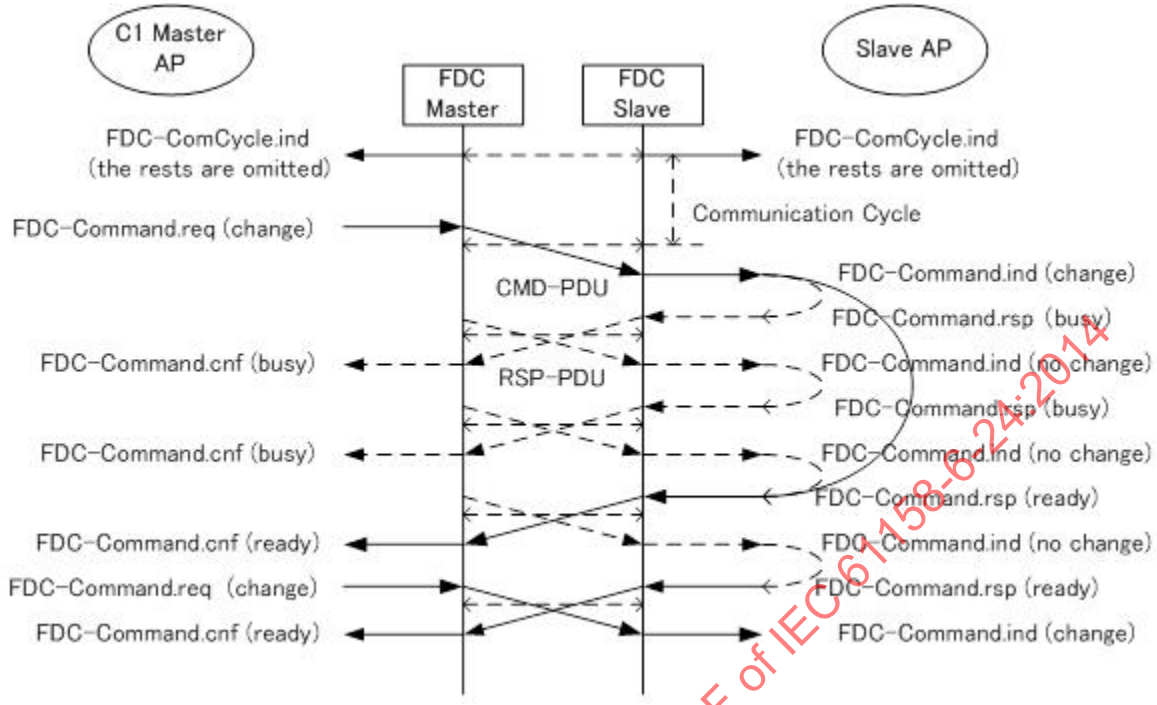
8.2.2.3 État de communication asynchrone (AsyncConnected)

Même à l'état de transmission asynchrone, le processus de communication du FDC ASE doit être périodiquement synchronisé avec le cycle de communication, car il est en mode de communication cyclique. Dans ce cas, il convient que l'AP maître et l'AP esclave n'utilisent pas de champ _WDT (horloge de surveillance: WDT) sur chaque SDU pour surveiller l'activité de synchronisation.

Par conséquent, l'événement périodique constant avec FDC-ComCycle.ind peut être notifié au processus utilisateur dans chaque cycle de communication. Toutefois, on peut ne pas toujours synchroniser le processus utilisateur avec l'événement.

Dans cet état, le FDC ASE doit doter le processus utilisateur de commandes de type asynchrone uniquement. Par conséquent, l'AP maître peut transférer une nouvelle commande uniquement après avoir confirmé la réponse de fin de processus correspondant à la commande de transfert.

La Figure 16 présente le chronogramme de l'état de communication asynchrone.



Légende

Anglais	Français
C1 Master AP	AP maître C1
FDC Master	Maître FDC
FDC Slave	Esclave FDC
Slave AP	AP esclave
(the rests are omitted)	(le reste est ignoré)
Communication Cycle	Cycle de communication
(change)	Changement
(busy)	Occupé
(ready)	Prêt
(no change)	Aucun changement

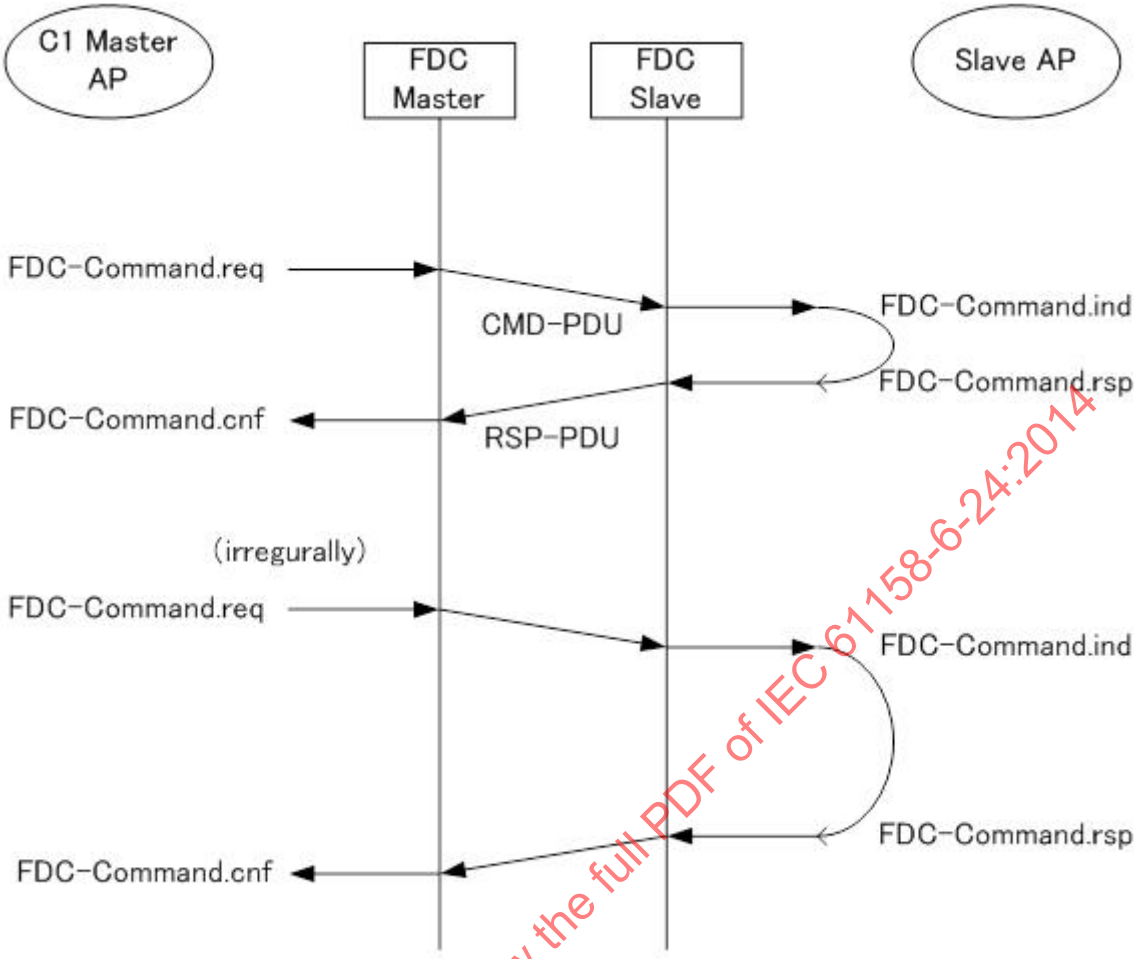
Figure 16 – Communication de commande asynchrone à l'état async

8.2.3 Mode de communication déclenchée par les événements

En mode de communication déclenchée par les événements, le FDC ASE peut ne comporter aucun cycle de communication et le processus de communication ne doit pas être exécuté de manière cyclique. L'horloge réseau ne peut pas fonctionner non plus.

Dans ce mode, le FDC ASE ne fournit aucun service lié à la commande de type synchrone, et le processus utilisateur peut uniquement utiliser les commandes de type asynchrone. L'AP maître peut demander le transfert non périodique d'une commande à chaque fois que la transaction de commande précédente est déjà terminée.

La Figure 17 présente le chronogramme du mode de communication déclenchée par les événements.



Légende

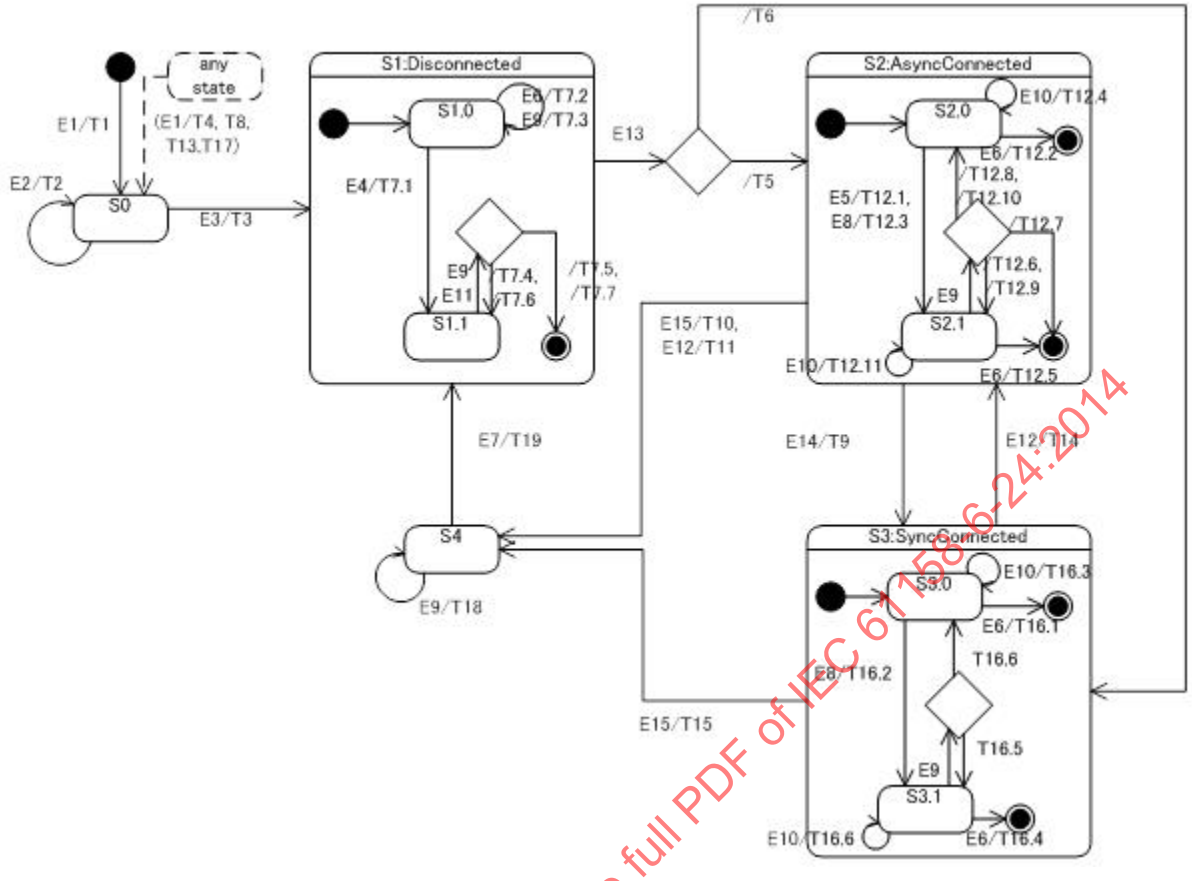
Anglais	Français
C1 Master AP	AP maître C1
FDC Master	Maître FDC
FDC Slave	Esclave FDC
Slave AP	AP esclave
irregurally	irrégulièrement

Figure 17 – Communication déclenchée par les événements

8.2.4 Machine protocolaire maître (FDCPM-M)

8.2.4.1 Descriptions des états

La Figure 18 présente le schéma d'états FDCPM-M et le Tableau 9 décrit chaque état de la FDCPM-M.



Légende

Anglais	Français
(any state)	(n'importe quel état)
Disconnected	Déconnecté

Figure 18 – Schéma d'états de la FDCPM-M

Tableau 9 – Descriptions des états de la FDCPM-M

S#	État	Sous-état	Description
S0	Désactivé	-	État lorsque la classe maître de FDC ASE (maître FDC) vient d'être instanciée vers un objet tel que cette PM La PM attend de terminer la création d'un contexte AP et de recevoir un Enable.requet de la part du FSM ASE. --- Les valeurs Entry/Initial sont définies dans les attributs de l'objet maître.
S1	Déconnecté	-	État lorsque la connexion est libérée et que la PM attend un Connect.request d'un utilisateur FAL Dans la couche inférieure, une communication a été démarrée. Seules les commandes de contrôle de connexion (CONNECT, DISCONNECT) et la commande NOP peuvent être transférées dans cet état. Même si la couche DL s'exécute en mode de communication cyclique, le cycle de communication n'a pas été notifié dans cet état car aucune connexion n'a été établie.

S#	État	Sous-état	Description
S1.0		Repos	Sous-état lorsque la PM attend de transférer une CMD-PDU d'un utilisateur FAL
S1.1		WaitCMDComplete	Sous-état lorsque la PM attend une réponse de fin de commande provenant de l'AP esclave homologue correspondant
S2	AsyncConnected	-	État de fonctionnement normal permettant de contrôler un AP esclave ou une fonction d'un appareil de terrain grâce à l'esclave FDC homologue à l'aide d'une commande de type asynchrone Une connexion à l'esclave FDC est établie et un événement de cycle de communication notifié par l'AR ASE. Cet événement est transféré en tant que FDC-ComCycle.ind vers l'AP maître C1. Toutes les commandes de type asynchrone peuvent être transférées dans cet état. Aucune commande de type synchrone ne peut être transférée.
S2.0		Repos	Sous-état lorsque la PM attend une demande pour transférer une CMD-PDU d'un utilisateur FAL
S2.1		WaitCMDComplete	Sous-état lorsque la PM attend une réponse de fin de commande provenant de l'AP esclave homologue correspondant
S3	SyncConnected	-	État de fonctionnement présentant la contrainte de temps la plus importante permettant de contrôler un AP esclave ou une fonction d'un appareil de terrain grâce à l'esclave FDC homologue à l'aide de commandes synchrones et asynchrones Une connexion à l'esclave FDC a été établie et un événement de cycle de communication notifié par l'AR ASE. Cet événement est transféré en tant que FDC-ComCycle.ind vers l'AP maître C1. Le compteur WDT de chaque CMD-PDU doit fonctionner de manière à surveiller l'état de l'activité de synchronisation de l'AP maître C1. De même pour le compteur RWDT des RSP-PDU pour l'AP esclave.
S3.0		Repos	Sous-état lorsque la PM attend de transférer une CMD-PDU d'un utilisateur FAL
S3.1		WaitCMDComplete	Sous-état lorsque la PM attend une réponse de fin de commande provenant de l'AP esclave homologue correspondant
S4	Déconnexion	-	État lorsque la connexion est libérée en raison d'erreurs de communication récurrentes ou parce que l'utilisateur FAL le demande. La PM attend afin d'éviter un dysfonctionnement du système en guise de mécanisme de verrouillage, puis revient à l'état S1 suite à la réception d'une demande de reprise de la part de l'utilisateur (ResumeCycle.req).

8.2.4.2 Événements déclencheurs

Le Tableau 10 répertorie chaque événement déclencheur de la FDCPM-M.

Tableau 10 – Descriptions des événements déclencheurs de la FDCPM-M

E#	Événement déclencheur: primitive ou condition	Source de l'événement	Paramètres associés	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	Utilisateur FAL	Update, CONNECT-CMD-	

E#	Événement déclencheur: primitive ou condition	Source de l'événement	Paramètres associés	Description
		(AP maître C1)	SDU	
E5	FDC-SyncSet.req	Utilisateur FAL (AP maître C1)	Updata, SYNC_SET-CMD-SDU	
E6	FDC-Disconnect.req	Utilisateur FAL (AP maître C1)	DISCONNECT-CMD-SDU	
E7	FDC-ResumeCycle.req	Utilisateur FAL (AP maître C1)		
E8	FDC-Command.req	Utilisateur FAL (AP maître C1)	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	Erreur détectée	Cet objet		Voir 8.2.7 et les notes du Tableau 11 Les défaillances WDT sont détectées deux fois en série à l'état SyncConnect. Par ailleurs, les défaillances sont détectées en continu à l'état AsyncConnect.
E13	EventConnectRSP	Cet objet (sous-machine)	état Unsigned16 rsdu _CONNECT-RSP-PDU	Notification de la sous-machine
E14	EventSyncSetRSP	Cet objet (sous-machine)	état Unsigned16 rsdu _SYNC_SET-RSP-PDU	Notification de la sous-machine
E15	EventDisconnectCMD	Cet objet (sous-machine)	csdu _DISCONNECT-CMD-PDU	Notification de la sous-machine

8.2.4.3 Descriptions des actions aux transitions d'état

Le Tableau 11 décrit les transitions d'état du diagramme d'états principal de la FDCPM-M. De plus, le Tableau 12 décrit les transitions d'état de la sous-machine de la FDCM-M.

Tableau 11 – Transitions du diagramme d'états principal de la FDCPM-M

T#	État source	Événement (arguments) [conditions] / action	État cible
T1	(n'importe quel état)	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC master --*/;	S0:Désactivé
T2	S0:Désactivé	E2:FDC-Open.req / /*-- Initializing the FDC master --*/; [a]	S0:Désactivé
T3	S0:Désactivé	E3:FDC-Enable.req (communicationMode) /* Notifiying the DLL's communication mode, initialized into whether the cyclic mode or the event-driven mode. */ / TransMode = communicationMode;	S1:Déconnecté
T4	S1:Déconnecté	E1:FDC-Reset.req /	S0:Désactivé

T#	État source	Événement (arguments) [conditions] / action	État cible
		/*-- Clearing all attributes of FDC master --*/;	
T5	S1:Déconnecté	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:Déconnecté	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:Déconnecté	The other events / /*-- state-transition in the submachine --*/;	S1:Déconnecté
T8	S2:AsyncConnected	E1:FDC-Reset.req / /*-- Clearing all attributes of FDC master --*/;	S0:Désactivé
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:Déconnexion
T11	S2:AsyncConnected	E12: Errors occur continuously [a, b, c]. / /*-- no-operation --*/;	S4:Déconnexion
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:Désactivé
T14	S3:SyncConnected	E12: Errors occur serially twice. [b] / /*-- no-operation --*/;	S2:AsyncConnected
T15	S3:SyncConnected	E15:EventDisconnectCMD (csdu) / /*-- no-operation --*/;	S4:Déconnexion
T16	S3:SyncConnected	The other events / /*-- state-transition in the submachine --*/;	S3:SyncConnected
T17	S4:Déconnexion	E1:FDC-Reset.req / /*-- Clearing all attributes of the FDC master --*/	S1:Désactivé
T18	S4:Déconnexion	E9:FDC-DataExchange.req (rsdu) / csdu.cmd = disconnect; FDC-DataExchange.cnf (csdu);	S4:Déconnexion
T19	S4:Déconnexion	E7:FDC-ResumeCycle.req / /*-- Clearing internal information related to the communication cycle and alarm information --*/;	S2:Déconnecté

a Le processus détaillé dépend de la mise en œuvre du système.

b Les implémenteurs utilisent un mécanisme de comptage ou de gestion des erreurs.

c La limite de comptage de la durée d'erreur est laissée à la discrétion de l'implémenteur.

Tableau 12 – Transitions de la sous-machine de la FDCPM-M

T#	État source	Événement (arguments) [conditions / action	État cible
T7.1	S1.0:Repos	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:Repos	E6:FDC-Disconnect.req (csdu) / LastCMD-SDU = csdu; If (TransMode == EventDriven) AR-SendCommand.req (csdu); else /*-- no-operation --*/;	S1.0: Repos
T7.3	S1.0:Repos	E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf (csdu);	S1.0:Repos
T7.4	S1.1: WaitCMDComplete	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: WaitCMDComplete	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 --*/;	Sortie vers la machine principale
T7.6	S1.1: 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; AR-SendCommand.req (csdu); /* Comment: TransMode is EventDriven. */	S1.1: WaitCMDComplete
T7.7	S1.1: WaitCMDComplete	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 --*/;	Sortie vers la machine principale
T12.1	S2.0:Repos	E5:FDC-SyncSet.req (csdu) / LastCMD-SDU = csdu;	S2.1: WaitCMDComplete
T12.2	S2.0:Repos	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 --*/;	Sortie vers la machine principale
T12.3	S2.0:Repos	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:Repos	E10: AR-CycleEvent.ind / FDC-ComCycle.ind	S2.0: Repos
T12.5	S2.1: WaitCMDComplete	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 --*/;	Sortie vers la machine principale
T12.6	S2.1: WaitCMDComplete	E9:FDC-DataExchange.req (rsdu) [/*-- rsdu has no alarm --*/ && (rsdu.rcmd == LastCMD-SDU.cmd) && (status.cmdRdy !=	S2.1: WaitCMDComplete

T#	État source	Événement (arguments) [conditions / action	État cible
		1)] / LastRSP-SDU = rsdu; csdu = LastCMD-SDU; FDC-DataExchange.cnf (csdu) ;	
T12.7	S2.1: WaitCMDComplete	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 --*/;	Sortie vers la machine principale
T12.8	S2.1: WaitCMDComplete	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:Repos

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

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 ou Sortie vers la machine principale, si E12 a été émis.

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;
			S3.0:Repos ou Sortie vers la machine principale, si E12 a été émis.

T#	État source	Événement (arguments) [conditions / action	État cible

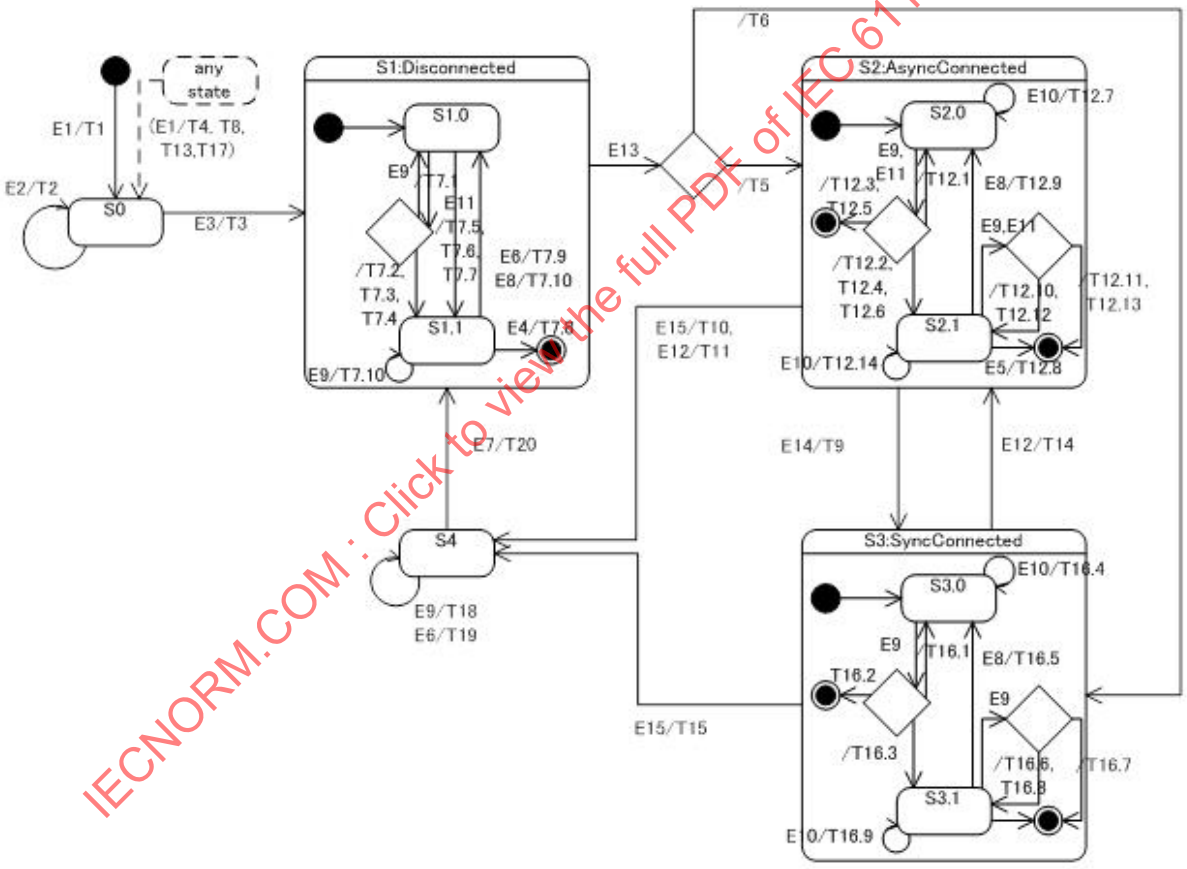
		csdu.wdt.mn = LastMN;
		csdu.wdt.sn = LastSN;
		FDC-DataExchange.cnf (csdu) ;
		FDC-Command.cnf (status, rsdu);}	

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

8.2.5 Machine protocolaire esclave (FDCPM-S)

8.2.5.1 Descriptions des états

La Figure 19 présente le schéma d'états FDCPM-S et le Tableau 13 chaque état de la FDCPM-S.



Légende

Anglais	Français
(any state)	(n'importe quel état)
Disconnected	Déconnecté

Figure 19 – Schéma d'états de la FDCPM-S