IEC 61158-6-17

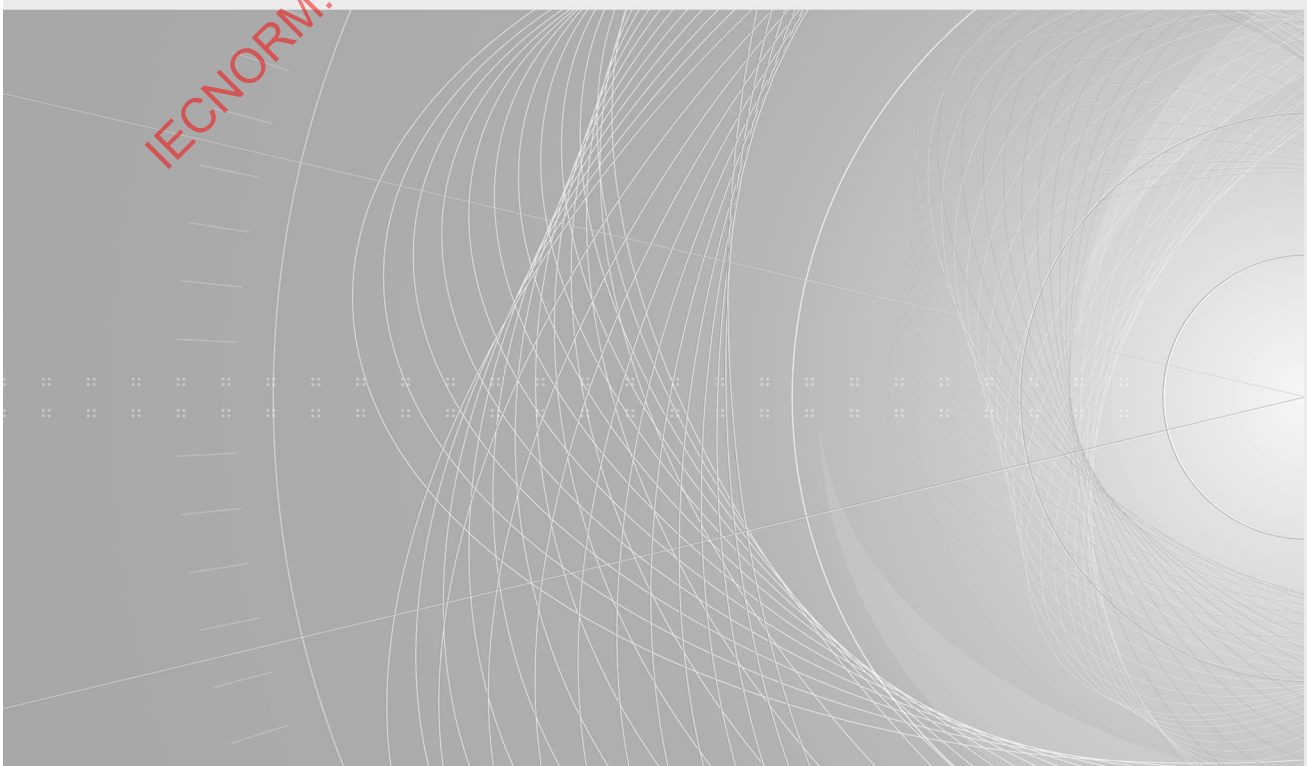
Edition 1.0 2007-12

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications –
Part 6-17: Application layer protocol specification – Type 17 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-17: Spécification de protocole de la couche d'application – Éléments de Type 17



IEC 61158-6-17:2007

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

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61158-6-17

Edition 1.0 2007-12

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications – Part 6-17: Application layer protocol specification – Type 17 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-17: Spécification de protocole de la couche d'application – Éléments de Type 17

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XB
CODE PRIX

ICS 25.040.40; 35.100.70

ISBN 978-2-8322-1025-3

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-17: Application layer protocol specification – Type 17 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 provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
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.

NOTE Use of some of the associated protocol types is restricted by their 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 particular data-link layer protocol type to be used with physical layer and application layer protocols in Type combinations as specified explicitly in the IEC 61784 series. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

IEC draws attention to the fact that it is claimed that compliance with this standard may involve the use of patents as follows, where the [xx] notation indicates the holder of the patent right:

Type 17:

PCT Application No. PCT/JP2004/011537 [YEC] Communication control method

PCT Application No. PCT/JP2004/011538 [YEC] Communication control method

IEC takes no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holders of these patent rights are registered with IEC. Information may be obtained from:

[YEC]: Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi, 180-8750 Tokyo,
180-8750 Tokyo,
Japan
Attention: Intellectual Property & Standardization Center

Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61158-6-17 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This first edition and its companion parts of the IEC 61158-6 subseries cancel and replace IEC 61158-6:2003. This edition of this part constitutes a technical addition. This part and its Type 17 companion parts also cancel and replace IEC/PAS 62405, published in 2005.

This edition of IEC 61158-6 includes the following significant changes from the previous edition:

a) deletion of the former Type 6 fieldbus for lack of market relevance;

b) addition of new types of fieldbuses;

c) partition of part 6 of the third edition into multiple parts numbered -6-2, -6-3, …

This bilingual version (2013-09) corresponds to the monolingual English version, published in 2007-12. The text of this standard is based on the following documents:

FDIS	Report on voting
65C/476/FDIS	65C/487/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.

The French version of this standard has not been voted upon.

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

The committee has decided that the contents of this publication will remain unchanged until the maintenance result 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.

NOTE The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

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

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/TR 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 implementors 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 6-17: Application layer protocol specification – Type 17 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 17 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 specifies interactions between remote applications and defines the externally visible behavior provided by the Type 17 fieldbus application layer in terms of

a) the formal abstract syntax defining the application layer protocol data units conveyed between communicating application entities;
b) the transfer syntax defining encoding rules that are applied to the application layer protocol data units;
c) the application context state machine defining the application service behavior visible between communicating application entities;
d) the application relationship state machines defining the communication behavior visible between communicating application entities.

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

1) define the wire-representation of the service primitives defined in IEC 61158-5-17, and
2) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the Type 17 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) 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 61158-5-17.

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

1.3 Conformance

This 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 reference

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

IEC 61158-5-17, *Industrial communication networks – Fieldbus specifications - Part 5-17: Application layer service definition – Type 17 elements*

ISO/IEC 7498 (all parts), *Information technology – Open Systems Interconnection – Basic Reference Model*

ISO/IEC 8824-2, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*

ISO/IEC 8825-1, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

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

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

3 Definitions

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

3.1 Terms and definitions

3.1.1 ISO/IEC 7498-1 terms

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

d) application entity
e) application protocol data unit
f) application service element

3.1.2 ISO/IEC 8824-2 terms

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

a) any type
b) bitstring type
c) Boolean type
d) choice type
e) false
f) integer type
g) null type
h) octetstring type

i) sequence of type
j) sequence type
k) simple type
l) structured type
m) tagged type
n) true
o) type
p) value

3.1.3 ISO/IEC 10731 terms

a) (N)-connection
b) (N)-entity
c) (N)-layer
d) (N)-service
e) (N)-service-access-point
f) confirm (primitive)
g) indication (primitive)
h) request (primitive)
i) response (primitive)

3.1.4 Other terms and definitions

3.1.4.1
application
function or data structure for which data is consumed or produced

3.1.4.2
application process
part of a distributed application on a network, which is located on one device and unambiguously addressed

3.1.4.3
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation

NOTE This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities

3.1.5
application relationship application service element
application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.1.5.1
application relationship endpoint
context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship

NOTE Each application process involved in the application relationship maintains its own application relationship endpoint.

3.1.5.2
attribute
description of an externally visible characteristic or feature of an object

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

3.1.5.3
behaviour
indication of how an object responds to particular eventss

3.1.5.4
bridge
intermediate equipment that connects two or more segments using a data-link layer relay function

3.1.5.5
channel
single physical or logical link of an input or output application object of a server to the process

3.1.5.6
class
a set of objects, all of which represent the same kind of system component

NOTE A class is a generalisation of an object; a template for defining variables and methods. All objects in a class are identical in form and behaviour, but usually contain different data in their attributes.

3.1.5.7
client
a) object which uses the services of another (server) object to perform a task

b) initiator of a message to which a server reacts

3.1.5.8
connection
logical binding between application objects that may be within the same or different devices

NOTE 1 Connections may be either point-to-point or multipoint.

NOTE 2 The logical link between sink and source of attributes and services at different custom interfaces of RT-Auto ASEs is referred to as interconnection. There is a distinction between data and event interconnections. The logical link and the data flow between sink and source of automation data items is referred to as data interconnection. The logical link and the data flow between sink (method) and source (event) of operational services is referred to as event interconnection.

3.1.5.9
connection point
buffer which is represented as a subinstance of an Assembly object

3.1.5.10
conveyance path
unidirectional flow of APDUs across an application relationship

3.1.5.11
dedicated AR
AR used directly by the FAL User

NOTE On Dedicated ARs, only the FAL Header and the user data are transferred.

3.1.5.12
device
physical hardware connected to the link

NOTE A device may contain more than one node.

3.1.5.13
domain
part of the RTE network consisting of one or two subnetwork(s)

NOTE Two subnetworks are required to compose a dual-redundant RTE network, and each end node in the domain is connected to both of the subnetworks.

3.1.5.14
domain master
station which performs diagnosis of routes to all other domains, distribution of network time to nodes inside the domain, acquisition of absolute time from the network time master and notification of status of the domain

3.1.5.15
domain number
numeric identifier which indicates a domain

3.1.5.16
end node
producing or consuming node

3.1.5.17
endpoint
one of the communicating entities involved in a connection

3.1.5.18
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.1.5.19
error class
general grouping for related error definitions and corresponding error codes

3.1.5.20
external bridge
bridge to which neither internal bridges nor RTE stations are connected directly

3.1.5.21
event
an instance of a change of conditions

3.1.5.22
group
a) <general> a general term for a collection of objects. Specific uses:

b) <addressing> when describing an address, an address that identifies more than one entity

3.1.5.23
interface
a) shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics as appropriate

b) collection of FAL class attributes and services that represents a specific view on the FAL class

3.1.5.24
interface port
physical connection point of an end node, which has an independent DL-address

3.1.5.25
internal bridge
bridge to which no routers, external bridges or nodes non-compliant with this specification are connected directly

3.1.5.26
invocation
act of using a service or other resource of an application process

NOTE Each invocation represents a separate thread of control that may be described by its context. Once the service completes, or use of the resource is released, the invocation ceases to exist. For service invocations, a service that has been initiated but not yet completed is referred to as an outstanding service invocation. Also for service invocations, an Invoke ID may be used to unambiguously identify the service invocation and differentiate it from other outstanding service invocations.

3.1.5.27
junction bridge
bridge to which at least one router, external bridge or node non-compliant with this specification, and to which at least one internal bridge or RTE station is connected

3.1.5.28
link
physical communication channel between two nodes

3.1.5.29
method
<object> a synonym for an operational service which is provided by the server ASE and invoked by a client

3.1.5.30
network
a set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.1.5.31
network time master
station which distributes network time to domain masters

3.1.5.32
node
single DL-entity as it appears on one local link

3.1.5.33
non-redundant interface node
node whch has a single interface port

3.1.5.34
non-redundant station
station that consists of a single end node

NOTE "non-redundant station" is synonymous with "end node".

3.1.5.35
object
abstract representation of a particular component within a device, usually a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behaviour

3.1.5.36
originator
client responsible for establishing a connection path to the target

3.1.5.37
path
logical communication channel between two nodes, which consists of one or two link(s)

3.1.5.38
peer
role of an AR endpoint in which it is capable of acting as both client and server

3.1.5.39
producer
node that is responsible for sending data

3.1.5.40
provider
source of a data connection

3.1.5.41
publisher
role of an AR endpoint that transmits APDUs onto the fieldbus for consumption by one or more subscribers

NOTE A publisher may not be aware of the identity or the number of subscribers and it may publish its APDUs using a dedicated AR.

3.1.5.42
redundant interface node
node with two interface ports one of which is connected to a primary network, while the other is connected to a secondary network

3.1.5.43
redundant station
station that consists of a pair of end nodes

NOTE Each end node of a redundant station has the same station number, but has a different DL-address.

3.1.5.44
resource
a processing or information capability of a subsystem

3.1.5.45
RTE station
station compliant with this specification

3.1.5.46
route
logical communication channel between two communication end nodes

3.1.5.47
router
intermediate equipment that connects two or more subnetworks using a network layer relay function

3.1.5.48
segment
communication channel that connects two nodes directly without intervening bridges

3.1.5.49
server
a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request

b) object which provides services to another (client) object

3.1.5.50
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.1.5.51
station
end node or a pair of end nodes that perform a specific application function

3.1.5.52
station number
numeric identifier which indicates a RTE station

3.1.5.53
subnetwork
part of a network that does not contain any routers. A subnetwork consists of end nodes, bridges and segments

NOTE Every end node included in a subnetwork has the same IP network address.

3.1.5.54
subscriber
role of an AREP in which it receives APDUs produced by a publisher

3.2 Abbreviations and symbols

3.2.1 ISO/IEC 10731 abbreviations

ASE application-service-element
OSI Open Systems Interconnection

3.2.2 ISO/IEC 7498-1 abbreviations and symbols

DL- Data-link layer (as a prefix)
DLL DL-layer
DLM DL-management
DLS DL-service
DLSAP DL-service-access-point
DLSDU DL-service-data-unit

3.2.3 IEC 61158-5-17 abbreviations and symbols

AE application entity
AL application layer
AP application process
APDU application protocol data unit
AR application relationship
AREP application relationship endpoint
ASN.1 abstract syntax notation one
BCD binary coded decimal
Cnf confirmation
cnf confirmation primitive
Ev_ prefix for data types defined for event ASE
FAL fieldbus application layer
Gn_ prefix for data types defined for general use

ID identifier
IEC International Electrotechnical Commission
Ind indication
ind indication primitive
IP Internet protocol
ISO International Organization for Standardization
lsb least significant bit
msb most significant bit
PDU protocol data unit
Req request
req request primitive
Rsp response
rsp response primitive
SAP service access point
SDU service data unit

3.2.4 Other abbreviations and symbols

ARPM application relationship protocol machine
FSPM FAL service protocol machine
MSU-AR multipoint network-scheduled unconfirmed publisher/subscriber AREP
MTU-AR multipoint user-triggered unconfirmed publisher/subscriber AREP
PSU-AR point-to-point network-scheduled unconfirmed client/server AREP
PTC-AR point-to-point user-triggered confirmed client/server AREP
PTU-AR point-to-point user-triggered unconfirmed client/server AREP

3.3 Conventions

3.3.1 General conventions

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

This standard uses the descriptive conventions given in IEC 61158-5 subseries for FAL service definitions.

3.3.2 Conventions for APDU abstract syntax definitions

This standard uses the descriptive conventions given in ISO/IEC 8824-2 for APDU definitions.

3.3.3 Conventions for APDU transfer syntax definitions

This standard uses the descriptive conventions given in ISO/IEC 8825-1 for transfer syntax definitions.

3.3.4 Conventions for AE state machine definitions

The conventions used for AE state machine definitions are described in Table 1.

Table 1 – Conventions used for AE state machine definitions

No.	Current state	Event / condition => action	Next state
Name of this transition	The current state to which this state transition applies	Events or conditions that trigger this state transition. => The actions that are taken when the above events or conditions are met. The actions are always indented below events or conditions	The next state after the actions in this transition are taken

The conventions used in the descriptions for the events, conditions and actions are as follows:

:= The value of an item on the left is replaced by the 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.

xxx Parameter name.

Example:

Identifier := reason

means value of the 'reason' parameter is assigned to the parameter called 'Identifier.'

"xxx" Indicates fixed value.

Example:

Identifier := "abc"

means value "abc" is assigned to a parameter named 'Identifier.'

= A logical condition to indicate an item on the left is equal to an item on the right.

< A logical condition to indicate an item on the left is less than the item on the right.

> A logical condition to indicate an item on the left is greater than the item on the right.

<> A logical condition to indicate an item on the left is not equal to an item on the right.

&& Logical "AND"

|| Logical "OR"

The sequence of actions and the alternative actions can be executed using the following reserved words.

for

endfor

if

else

elseif

The following shows examples of description using the reserved words.

Example 1:

```
for (Identifier := start_value to end_value)
        actions
endfor
```

Example 2:

```
If (condition)
        actions
else
        actions
endif
```

4 Abstract syntax description

4.1 FAL PDU abstract syntax

4.1.1 Top level definition

```
FalArPDU ::=
        ConfirmedSend-CommandPDU
      || ConfirmedSend-ResponsePDU
      || UnconfirmedSend-CommandPDU
```

4.1.2 FalArHeader

```
FalArHeader ::= Unsigned8{
      -- bit 8-7        ProtocolVersion
      -- bit 6-4        ProtocolIdentifier
      -- bit 3-1        PDUIdentifier
}
```

4.1.3 Confirmed send service

```
ConfirmedSend-CommandPDU ::= SEQUENCE {
      FalArHeader,
      ServiceType
      InvokeID,
      ConfirmedServiceRequest
}

ConfirmedSend-ResponsePDU ::= SEQUENCE {
      FalArHeader,
      ServiceType
      InvokeID,
      ConfirmedServiceResponse
}
```

4.1.4 Unconfirmed send service

```
UnconfirmedSend-CommandPDU ::= SEQUENCE {
      FalArHeader,
      ServiceType
      InvokeID,
      UnconfirmedServiceRequest
}
```

4.2 Abstract syntax of PDU body

4.2.1 ConfirmedServiceRequest PDUs

```
ConfirmedServiceRequest ::= CHOICE {
      Read-Request              [0]   IMPLICIT   Read-RequestPDU,
      Write-Request             [1]   IMPLICIT   Write-RequestPDU,
      DownLoad-Request          [2]   IMPLICIT   DownLoad-RequestPDU,
      UpLoad-Request            [3]   IMPLICIT   UpLoad-RequestPDU,
      Start-Request             [4]   IMPLICIT   Start-RequestPDU,
      Stop-Request              [5]   IMPLICIT   Stop-RequestPDU,
      Resume- Request           [6]   IMPLICIT   Resume-RequestPDU,
      DelayCheck-Request        [7]   IMPLICIT   Time- RequestPDU,
}
```

4.2.2 ConfirmedServiceResponse PDUs

```
ConfirmedServiceResponse ::= CHOICE {
        Read-Response                   [0]     IMPLICIT    Read-ResponsePDU,
        Write-Response                  [1]     IMPLICIT    Write-ResponsePDU,
        DownLoad-Response               [2]     IMPLICIT    DownLoad-ResponsePDU,
        UpLoad-Response                 [3]     IMPLICIT    UpLoad-ResponsePDU,
        Start-Response                  [4]     IMPLICIT    Start-ResponsePDU,
        Stop-Response                   [5]     IMPLICIT    Stop-ResponsePDU,
        Resume-Response                 [6]     IMPLICIT    Resume-ResponsePDU,
        DelayCheck-Response             [7]     IMPLICIT    Time-ResponsePDU
}
```

4.2.3 Unconfirmed PDUs

```
UnconfirmedServiceRequest ::= CHOICE {
        InformationReport-Request       [0]     IMPLICIT    InformationReport-RequestPDU,
        EventNotification-Request       [1]     IMPLICIT    EventNotification-RequestPDU,
        EventRecovery-Request           [2]     IMPLICIT    EventRecovery-RequestPDU,
        TimeDistribution-Request        [3]     IMPLICIT    TimeDistribute-RequestPDU,
        SetTime-Request                 [4]     IMPLICIT    SetTime-RequestPDU,
        InDiag-Request                  [5]     IMPLICIT    InDiag-RequestPDU,
        ExDiag-Request                  [6]     IMPLICIT    ExDiag-RequestPDU,
        StationStatusReport-Request     [7]     IMPLICIT    StationStatusReport-RequestPDU,
        DomainStatusReport-Request      [8]     IMPLICIT    DomainStatusReport-RequestPDU
}
```

4.2.4 Error information

4.2.4.1 Error type

```
ErrorType ::= SEQUENCE {
        errorClass                      [0]     IMPLICIT    ErrorClass,
        additionalCode                  [1]     IMPLICIT    Integer16 OPTIONAL,
        additionalDescription           [2]     IMPLICIT    VisibleString OPTIONAL,
        additionalInfo                  [3]     IMPLICIT    ANY OPTIONAL
}
```

4.2.4.2 Error class

```
ErrorClass ::= CHOICE {
        noError                         [0]   IMPLICIT    Integer8 {
                                                  normal                          (0),
                                                  other                           (1)
        }
        applicationReference            [1]   IMPLICIT    Integer8 {
                                                  other                           (0),
                                                  application-unreachable         (1),
                                                  application-reference-invalid   (2),
                                                  context-unsupported             (3)
        }
        definition                      [2]   IMPLICIT    Integer8 {
                                                  other                           (0),
                                                  object-undefined                (1),
                                                  object-attributes-inconsistent  (2),
                                                  name-already-exists             (3),
                                                  type-unsupported                (4),
                                                  type-inconsistent               (5)
        }
        resource                        [3]   IMPLICIT    Integer8 {
                                                  other                           (0),
                                                  memory-unavailable              (1)
        }
        service                         [4]   IMPLICIT    Integer8 {
                                                  other                           (0),
                                                  object-state-conflict           (1),
                                                  pdu-size                        (2),
                                                  object-constraint-conflict      (3),
                                                  parameter-inconsistent          (4),
                                                  illegal-parameter               (5)
        }
        access                          [5]   IMPLICIT    Integer8 {
                                                  other                           (0),
                                                  object-invalidated              (1),
                                                  hardware-fault                  (2),
                                                  object-access-denied            (3),
                                                  invalid-address                 (4),
                                                  object-attribute-inconsistent   (5),
                                                  object-access-unsupported       (6),
                                                  object-non-existent             (7),
                                                  type-conflict                   (8),
                                                  named-access-unsupported        (9),
                                                  access-to-element-unsupported   (10)
        }
        conclude                        [6]   IMPLICIT    Integer8 {
                                                  other                           (0)
        }
        other                           [7]   IMPLICIT    Integer8 {
                                                  other                           (0)
        }
}
```

4.3 PDUs for ASEs

4.3.1 PDUs for Variable ASE

4.3.1.1 Read service PDUs

```
Read-RequestPDU ::= SEQUENCE {
        objectSpecifier CHOICE{
              variableSpecifier                 Gn_KeyAttribute,
              variableListSpecifier             Gn_KeyAttribute,
              listOfvariable                    SEQUENCE OF Gn_KeyAttribute
        }
        optionalParameters              [0]   IMPLICIT    ANY OPTIONAL
}
```

```
Read-ResponsePDU ::= SEQUENCE {
        result CHOICE{
              accessStatus                [0]    IMPLICIT    ErrorType,
              listOfAccessStatus          [1]    IMPLICIT    SEQUENCE OF ErrorType
        }
        value CHOICE{
              data                        [0]    IMPLICIT    ANY,
              listOfData                  [1]    IMPLICIT    SEQUENCE OF ANY
        }
        variableType CHOICE{
              dataType                    [0]    IMPLICIT    Gn_FullyNestedTypeDescription OPTIONAL,
              listOfDataType              [1]    IMPLICIT    SEQUENCE OF Gn_FullyNestedTypeDescription
                                                             OPTIONAL
        }
        optionalParameters                [0]    IMPLICIT    ANY OPTIONAL
}
```

4.3.1.2 Write service PDUs

```
Write-RequestPDU ::= SEQUENCE {
        objectSpecifier CHOICE{
              variableSpecifier                              Gn_KeyAttribute,
              variableListSpecifier                          Gn_KeyAttribute,
              listOfVariable                                 SEQUENCE OF Gn_KeyAttribute
        }
        variableType CHOICE{
              dataType                    [0]    IMPLICIT    Gn_FullyNestedTypeDescription OPTIONAL,
              listOfDataType              [1]    IMPLICIT    SEQUENCE OF Gn_FullyNestedTypeDescription
                                                             OPTIONAL
        }
        value CHOICE{
              data                        [0]    IMPLICIT    ANY,
              listOfData                  [1]    IMPLICIT    SEQUENCE OF ANY
        }
        optionalParameters                [0]    IMPLICIT    ANY OPTIONAL
}

Write-ResponsePDU ::= SEQUENCE {
        result CHOICE{
              accessStatus                [0]    IMPLICIT    ErrorType,
              listOfAccessStatus          [1]    IMPLICIT    SEQUENCE OF ErrorType
        }
        optionalParameters                [0]    IMPLICIT    ANY OPTIONAL
}
```

4.3.1.3 Information Report service PDUs

```
InformationReport-RequestPDU::= SEQUENCE {
        ListOfVariableSpecifier CHOICE {
              variableListSpecifier                          Gn_KeyAttribute,
              listOfVariable                                 SEQUENCE OF Gn_KeyAttribute
        },
        listOfDataType                    [1]    IMPLICIT    SEQUENCE OF Gn_FullyNestedTypeDescription
                                                             OPTIONAL
        listOfData                        [2]    IMPLICIT    SEQUENCE OF ANY
        optionalParameters                [3]    IMPLICIT    ANY OPTIONAL
}
```

4.3.2 PDUs for Event ASE

4.3.2.1 Event Notification service

```
EventNotification-RequestPDU ::= SEQUENCE {
      eventNotifierID                            IMPLICIT    Gn_ KeyAttribute,,
      notificvationSequenceNumber       [1]      IMPLICIT    Ev_SequenceNumber,
      listOfEvent                       [2]      IMPLICIT    SEQUENCE OF Ev_EventData,
      Notification Time                 [3]      IMPLICIT    Ev_TimeTag OPTIONAL
      optionalParameters                [4]      IMPLICIT    ANY OPTIONAL
}
```

4.3.2.2 Notification Recovery service

```
EventRecovery-RequestPDU ::= SEQUENCE {
        eventNotifierID                         IMPLICIT    Gn_ KeyAttribute,,
        sequenceNumber                  [1]     IMPLICIT    Ev_SequenceNumber OPTIONAL
}
```

4.3.3 PDUs for Load region ASE

4.3.3.1 Download service

```
DownLoad-RequestPDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        segmentIdentifier               [1]     IMPLICIT    ANY,
        loadData                        [2]     IMPLICIT    octetString,
}


DownLoad-ResponsePDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        result                          [1]     IMPLICIT    ErrorType,
}
```

4.3.3.2 Upload service

```
UpLoad-RequestPDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        segmentIdentifier               [1]     IMPLICIT    ANY,
}


UpLoad-ResponsePDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        result                          [1]     IMPLICIT    ErrorType,
        loadData                        [2]     IMPLICIT    octetString,
}
```

4.3.4 PDUs for Function Invocation ASE

4.3.4.1 Start service

```
Start-RequestPDU ::= SEQUENCE {
        keyAttribute                                        Gn_KeyAttribute,
        optionalParameters              [1]     IMPLICIT    ANY OPTIONAL
}


Start-ResponsePDU ::= ErrorType
```

4.3.4.2 Stop service

```
Stop-RequestPDU ::= SEQUENCE {
        keyAttribute                                        Gn_KeyAttribute,
        optionalParameters              [1]     IMPLICIT    ANY OPTIONAL
}


Stop-ResponsePDU ::= ErrorType
```

4.3.4.3 Resume services

```
Resume-RequestPDU ::= SEQUENCE {
        keyAttribute                                        Gn_KeyAttribute,
        optionalParameters              [1]     IMPLICIT    ANY OPTIONAL
}


Resume-ResponsePDU ::= ErrorType
```

4.3.5 PDUs for Time ASE

4.3.5.1 Time service

```
Time-RequestPDU ::= Time-PDU

Time-ResponsePDU ::= Time-PDU

TimeDistribute-RequestPDU ::= Time-PDU

Time-PDU ::= SEQUENCE {
        timeControl             [0]   IMPLICIT   Tm_TimeControl,
        Stratum                 [1]   IMPLICIT   Unsigned8
        PollInterval            [2]   IMPLICIT   Tm_TimeValue1,
        Precision               [3]   IMPLICIT   Tm_TimeValue1,
        rootDelay               [4]   IMPLICIT   Tm_TimeValue2,
        rootDispersion          [5]   IMPLICIT   Tm_TimeValue2,
        referenceIdentifier     [6]   IMPLICIT   Tm_ReferenceID,
        referenceTimestamp      [7]   IMPLICIT   Tm_Time,
        originateTimestamp      [8]   IMPLICIT   Tm_Time,
        receiveTimestamp        [9]   IMPLICIT   Tm_Time,
        transmitTimestamp       [10]  IMPLICIT   Tm_Time,
}

SetTime-RequestPDU ::= SEQUENCE {
        timeValue               [0]   IMPLICIT   Tm_Time,
        optionalParameters      [1]   IMPLICIT   ANY OPTIONAL
}
```

4.3.6 PDUs for Network Management ASE

4.3.6.1 Network Management service

```
InDiag-RequestPDU ::= SEQUENCE {
        nodeInformation         [0]   IMPLICIT   Nm_NodeInformation,
        nodeStatus              [1]   IMPLICIT   Nm_NodeStatus,
        nodePublicKey           [2]   IMPLICIT   Nm_PublicKey,
        llistOfPathStatus       [3]   IMPLICIT   Nm_ListOfPathStatus
}

ExDiag-RequestPDU ::= SEQUENCE {
        doaminInformation       [0]   IMPLICIT   Nm_DoaminInformation,
        domainStatus            [1]   IMPLICIT   Nm_DoaminStatus,
        domainPublicKey         [2]   IMPLICIT   Nm_PublicKey,
        masterPriority          [3]   IMPLICIT   Unsigned8,
        llistOfPathStatus       [4]   IMPLICIT   Nm_ListOfPathStatus,
        listOfNodeStatus        [5]   IMPLICIT   SEQUENCE OF Nm_NodeStatus
}

StationStatusReport-RequestPDU ::= SEQUENCE {
        nodeInformation         [0]   IMPLICIT   Nm_NodeInformation,
        nodeStatus              [1]   IMPLICIT   Nm_NodeStatus
}

DomainStatusReport-RequestPDU ::= SEQUENCE {
        doaminInformation       [0]   IMPLICIT   Nm_DoaminInformation,
        domainStatus            [1]   IMPLICIT   Nm_DomainStatus
}
```

4.4 Type definitions

4.4.1 Variable ASE types

There are no types special for the Variable ASE.

4.4.2 Event ASE types

```
Ev_SequenceNumber ::= Unsigned8

Ev_EventData ::= ANY

En_EventCount ::= Unsigned8

Ev_TimeTag ::= Unsigned16
```

4.4.3 Load Region ASE types

There are no types special for the Load Region ASE.

4.4.4 Function Invocation ASE types

There are no types special for the function Invocation ASE.

4.4.5 Time ASE types

```
Tm_TimeControl ::= BitString8 {
        -- bit 8,7      LeapIndidator
        -- bit 6-4      ProtocolVersion
        -- bit 3-1      TimeMode
}

Tm_TimeValue1 ::= Unsigned32        -- eight-bit signed integer, in seconds to the nearest power of two

Tm_TimeValue2 ::= Unsigned32        -- 32-bit signed fixed-point number, in seconds
                                    -- with fraction point between bits 15 and 16

Tm_ReferenceID ::= VisibleString4   -- identifies the particular reference source

Tm_Time ::= SEQUENCE{
    Seconds             [0]             Unsigned32
    SecondsFraction     [2]             Unsigned32
}
```

4.4.6 Network Management ASE types

```
Nm_NodeInformation ::= SEQUENCE {
    NodeIdentifier                  [0]   IMPLICIT   Nm_NodeIdentifier,
    NoOfInterfaces                  [1]   IMPLICIT   Integer8,
    InterfaceID                     [2]   IMPLICIT   Unsigned8,
    PerformanceClass SEQUENCE {
        MasterPriority              [11]  IMPLICIT   Unsigned8,
        TransmissionClass           [12]  IMPLICIT   Unsigned8,
        ResponseClass               [13]  IMPLICIT   Unsigned8,
        TimePrecisionLevel          [14]  IMPLICIT   Unsigned8,
    }
    configurationSUM                [4]   IMPLICIT   Unsigned32,
    localNodeTime                   [5]   IMPLICIT   Tm_Time,
    diagInterval                    [6]   IMPLICIT   BinaryTime2,
    stationCoefficeincy             [7]   IMPLICIT   Unsigned16
}

Nm_NodeStatus ::= BitString8 {
        -- bit 8        CPU-Status                  -- True: ready, False: not ready
        -- bit 7        communication-status        -- True: ready, False: not ready
        -- bit 6        reserved-status             -- True: reserved, False: not reserved
        -- bit 5        redundancy-status           -- True: on-service, False: stand-by
        -- bit 4        linkStatusOfnterfaceB       -- True: linked, False: not linked
        -- bit 3        linkStatusOfnterfaceA       -- True: linked, False: not linked
        -- bit 2        statusOfNetworkB            -- True: healthy, False: failed
        -- bit 1        statusOfNetworkA            -- True: healthy, False: failed
}
```

```
Nm_PublicKey ::= Unsigned64

Nm_ListOfPathStatus ::= CompactBooleanArray          -- True: healthy, False: failed

Nm_DoaminInformation ::= SEQUENCE {
        NodeIdentifier              [0]   IMPLICIT   Nm_NodeIdentifier,
        NoOfInterfaces              [1]   IMPLICIT   Integer8,
        InterfaceID                 [2]   IMPLICIT   Unsigned8,
        localNodeTime               [3]   IMPLICIT   Tm_Time,
        diagInterval                [4]   IMPLICIT   BinaryTime2,
}

Nm_DomainStatus ::= BitString8 {
        -- bit 8        statusOfNetworkB              -- True: healthy, False: failed
        -- bit 7        statusOfNetworkA              -- True: healthy, False: failed
        -- bit 6,5      StatusOfTimeSynchronization   -- 00: not synchronized
                                                      -- 01: synchronized with the domain time master
                                                      -- 10: synchronized with the network time master
                                                      -- 11: synchronized with the external time source
        -- bit 4-1      TimeGroup
}

Nm_NodeIdentifier ::= SEQUENCE {
        DomainNumber                [0]   IMPLICIT   Integer8,
        StationNumber               [1]   IMPLICIT   Integer8
}
```

4.4.7 General types

4.4.7.1 Gn_KeyAttribute

```
Gn_KeyAttribute ::= CHOICE {
-- When this type is specified, only the key attributes of the class referenced are valid.
        numericID                   [0]   IMPLICIT   Gn_NumericID,
        name                        [1]   IMPLICIT   Gn_Name,
        listName                    [2]   IMPLICIT   Gn_Name,
        numericAddress              [4]   IMPLICIT   Gn_NumericAddress,
        symbolicAddress             [5]   IMPLICIT   Gn_SymbolicAddress
}
```

4.4.7.2 Gn_Name

```
Gn_Name ::= octetString
```

4.4.7.3 Gn_NumericAddress

```
Gn_NumericAddress ::= SEQUENCE {
        startAddress            [0]   IMPLICIT   Unsigned32,   -- physical address of the starting location
        length                  [1]   IMPLICIT   Unsigned16   -- octet length of a memory block
}
```

4.4.7.4 Gn_NumericID

```
Gn_NumericID ::= Unsigned16          -- The values of this parameter are unique within an AP.
```

4.4.7.5 Gn_SymbolicAddress

```
Gn_SymbolicAddress ::= VisibleString
```

4.4.7.6 Gn_FullyNestedTypeDescription

```
Gn_FullyNestedTypeDescription ::= CHOICE {
        boolean                 [1]             Unsigned8,
        integer8                [2]             Unsigned8,
        integer16               [3]             Unsigned8,
        integer32               [4]             Unsigned8,
        unsigned8               [5]             Unsigned8,
        unsigned16              [6]             Unsigned8,
        unsigned32              [7]             Unsigned8,
        float32                 [8]             Unsigned8,
        float64                 [9]             Unsigned8,
        binaryDate              [10]            Unsigned8,
        timeOfDay               [11]            Unsigned8,
        timeDifference          [12]            Unsigned8,
        universalTime           [13]            Unsigned8,
        fieldbusTime            [14]            Unsigned8,
        time                    [15]            Unsigned8,
        bitstring8              [16]            Unsigned8,
        bitstring16             [17]            Unsigned8,
        bitstring32             [18]            Unsigned8,
        visiblestring1          [19]            Unsigned8,
        visiblestring2          [20]            Unsigned8,
        visiblestring4          [21]            Unsigned8,
        visiblestring8          [22]            Unsigned8,
        visiblestring16         [23]            Unsigned8,
        octetstring1            [24]            Unsigned8,
        octetstring2            [25]            Unsigned8,
        octetstring4            [26]            Unsigned8,
        octetstring8            [27]            Unsigned8,
        octetstring16           [28]            Unsigned8,
        bcd                     [29]            Unsigned8,
        iso10646char            [30]            Unsigned8,
        binarytime0             [31]            Unsigned8,
        binarytime1             [32]            Unsigned8,
        binarytime2             [33]            Unsigned8,
        binarytime3             [34]            Unsigned8,
        binarytime4             [35]            Unsigned8,
        binarytime5             [36]            Unsigned8,
        binarytime6             [37]            Unsigned8,
        binarytime7             [38]            Unsigned8,
        binarytime8             [39]            Unsigned8,
        binarytime9             [40]            Unsigned8,
        visiblestring           [41]            Unsigned8,
        octetstring             [42]            Unsigned8,
        bitstring               [43]            Unsigned8,
        compactBooleanArray     [44]            Unsigned8,
        compactBCDArray         [45]            Unsigned8,
        iso646string            [46]            Unsigned8,
        structure               [47]  IMPLICIT  SEQUENCE OF Gn_FullyNestedTypeDescription
}
```

4.5 Data types

4.5.1 Notation for the Boolean type

```
Boolean ::= BOOLEAN                     -- TRUE if the value is non-zero.
                                        -- FALSE if the value is zero.
```

4.5.2 Notation for the Integer type

```
Integer ::= INTEGER                     -- any integer
Integer8 ::= INTEGER (-128..+127)       -- range -27 <= i <= 27-1
Integer16 ::= INTEGER (-32768..+32767)  -- range -215 <= i <= 215-1
Integer32 ::= INTEGER                   -- range -231 <= i <= 231-1
```

4.5.3 Notation for the Unsigned type

```
Unsigned ::= INTEGER                    -- any non-negative integer
Unsigned8 ::= INTEGER (0..255)          -- range 0 <= i <= 28-1
Unsigned16 ::= INTEGER (0..65535)       -- range 0 <= i <= 216-1
Unsigned32 ::= INTEGER                  -- range 0 <= i <= 232-1
```

4.5.4 Notation for the Floating Point type

```
Floating32 ::= BIT STRING SIZE (4)          -- IEC-60559Single precision
Floating64 ::= BIT STRING SIZE ( 8)         -- IEC-60559Double precision
```

4.5.5 Notation for the BitString type

```
BitString ::= BIT STRING                    -- For generic use
BitString4 ::= BIT STRING SIZE (4)          -- Fixed four bits bitstring
BitString8 ::= BIT STRING SIZE (8)          -- Fixed eight bits bitstring
BitString16 ::= BIT STRING SIZE (16)        -- Fixed 16 bits bitstring
BitString32 ::= BIT STRING SIZE (32)        -- Fixed 32 two bits bitstring
```

4.5.6 Notation for the octetString type

```
octetString ::= OCTET STRING                -- For generic use
octetString2 ::= OCTET STRING SIZE (2)      -- Fixed two-octet octet string
octetString4 ::= OCTET STRING SIZE (4)      -- Fixed four-octet octet string
octetString6 ::= OCTET STRING SIZE (6)      -- Fixed six-octet octet string
octetString7 ::= OCTET STRING SIZE (7)      -- Fixed seven-octet octet string
octetString8 ::= OCTET STRING SIZE (8)      -- Fixed eight-octet octet string
octetString16 ::= OCTET STRING SIZE (16)    -- Fixed 16 octet octet string
```

4.5.7 Notation for VisibleString type

```
VisibleString2 ::= VisibleString SIZE (2)   -- Fixed two-octet visible string
VisibleString4 ::=VisibleString SIZE (4)    -- Fixed four-octet visible string
VisibleString8 ::= VisibleString SIZE (8)   -- Fixed eight-octet visible string
VisibleString16 ::= VisibleString SIZE (16) -- Fixed 16 octet visible string
```

4.5.8 Notation for the UNICODEString type

```
UNICODEString ::= UNICODEString             -- 16 bit character code set defined in ISO 10646.
```

4.5.9 Notation for Binary Time type

```
BinaryTime0 ::= BIT STRING SIZE (16)        -- 10 µs resolution
BinaryTime1 ::= BIT STRING SIZE (16)        -- 0.1 ms resolution
BinaryTime2 ::= BIT STRING SIZE (16)        -- 1 ms resolution
BinaryTime3 ::= BIT STRING SIZE (16)        -- 10 ms resolution
BinaryTime4 ::= BIT STRING SIZE (16)        -- 0.1 s resolution
BinaryTime5 ::= BIT STRING SIZE (16)        -- 1 s resolution
BinaryTime6 ::= BIT STRING SIZE (32)        -- 10 µs resolution
BinaryTime7 ::= BIT STRING SIZE (32)        -- 0.1 ms resolution
BinaryTime8 ::= BIT STRING SIZE (32)        -- 1 ms resolution
BinaryTime9 ::= BIT STRING SIZE (32)        -- 10 ms resolution
```

4.5.10 Notation for BCD type

```
BCD ::= Unsigned8 (0..9)                    -- Lower four bits are used to express one BCD value.
```

4.5.11 Notation for Compact Boolean Array type

```
CompactBooleanArray ::= BitString           -- Each zero bit representing Boolean value FALSE.
                                            -- Each one bit representing Boolean value TRUE.
                                            -- Unused bits, if any, shall be placed in bits 7-1 of the last octet.
```

4.5.12 Notation for Compact BCD Array type

```
CompactBCDArray ::= octetString             -- One BCD value is represented by four bits, an unused
                                            -- nibble, if any, shall be placed in bits 4-1 of the last octet,
                                            -- and shall be set to 1111F.
```

5 Transfer syntax

5.1 Overview of encoding

The FAL-PDUs encoded shall have a uniform format. The FAL-PDUs shall consist of two major parts, the “APDU Header” part and the “APDU Body” part as shown in Figure 1.

(1)	(1)	(1)	(n) --- octets
FalArHeader Field	Type Field	(InvokeID)	Service Specific Parameters
<---------- APDU Header ---------->			<---------- APDU Body ---------->

NOTE The presence of the InvokeID Field depends on the APDU type.

Figure 1 – APDU overview

To realize an efficient APDU while maintaining flexible encoding, different encoding rules are used for the APDU Header part and the APDU Body part.

NOTE The data-link layer service provides a DLSDU parameter that implies the length of the APDU. Thus, the APDU length information is not included in the APDU.

5.2 APDU header encoding

The APDU Header part is always present in all APDUs that conform to this standard. It consists of three fields: the FalArHeader Field, the Type Field, and the optional InvokeID Field.

They are shown in Figure 1.

5.2.1 Encoding of FalArHeader field

All the FAL PDUs shall have the common PDU-header called FalArHeader. The FalArHeader identifies abstract syntax, transfer syntax, and each of the PDUs. Table 2 defines how this header shall be used.

Table 2 – Encoding of FalArHeader field

Bit position of the FalArHeader			PDU type	Protocol version
8 7	6 5 4	3 2 1		
01	001	000	ConfirmedSend-CommandPDU	Version 1
01	001	100	ConfirmedSend-ResponsePDU	Version 1
01	010	000	UnconfirmedSend-CommandPDU	Version 1

NOTE All other code points are reserved for additional protocols and future revisions.

5.2.2 Encoding of Type field

a) The service type of an APDU is encoded in the Type Field that is always the second octet of the APDUs.

b) All bits of the Type Field are used to encode the service type.

1) The service types shall be encoded in bits 8 to 1 of the Type Field, with bit 8 the most significant bit and bit 1 the least significant bit. The range of service type shall be between 0 (zero) and 254, inclusive.

2) The value of 255 is reserved for future extensions to this specification.

3) The service type is specified in the abstract syntax as a positive integer value.

c) Figure 2 illustrates the encoding of the Type Field.

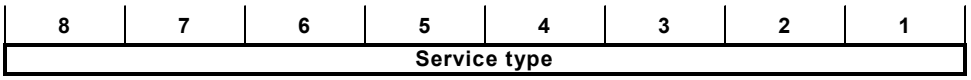


Figure 2 – Type field

5.2.3 Encoding of InvokeID Field

The InvokeID Field shall be present if it is indicated in the abstract syntax. Otherwise, this field shall not be present. If present, the InvokeID parameter supplied by a service primitive shall be placed in this field.

5.3 APDU body encoding

5.3.1 General

The FAL encoding rules are based on the terms and conventions defined in ISO/IEC 8825-1. The encoding consists of three components in the following order:

Identifier octet
Length octet(s)
Contents octet(s)

5.3.2 Identifier octet

The Identifier octet shall encode the tag defined in the FAL Abstract Syntax and shall consist of one octet.

It consists of the P/C flag and the Tag field as shown in Figure 3.

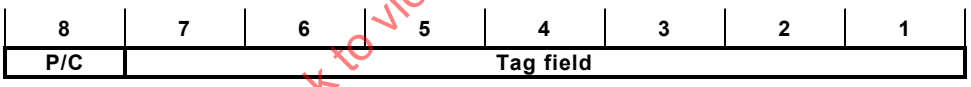


Figure 3 – Identifier octet

The P/C flag indicates that the Contents octet(s) is either a simple component (primitive types, such as Integer8), or a structured component (constructed, such as SEQUENCE, SEQUENCE OF types).

P/C Flag =0 means the Contents octet(s) is a simple component.
P/C Flag =1 means the Contents octet(s) is a structured component.

The Tag field identifies the semantics of the Contents octet(s).

5.3.3 Length octet(s)

The Length octet(s) shall consist of one or three octets.

a) If the value of the first Length octet is other than 255, there shall be no subsequent Length octet(s) and the first octet shall contain the value for the Length octet defined later.

b) If the value of the first Length octet is 255, there shall be two subsequent Length octet(s) that shall contain the values for the Length octets defined later. In this case, the length information of the Contents octet(s) shall be represented by the last two octets of the Length octets, where the most significant bit of the second of three Length octets shall be the most significant bit of the length value and the least significant bit of the third of the three Length octets shall be the least significant bit of the length value.

The sender shall have the option of using either the one-octet format or the three-octet format. For example, the three-octet format may be used to convey a length value of one.

The meaning of the Length octet(s) depends on the type of value being encoded. If the encoding of the Contents octet(s) is primitive, the Length octet(s) shall contain the number of octets in the Contents octets. If the encoding of the Contents octets is constructed, the Length octet(s) shall contain the number of the first-level components of the Contents octets.

Figure 4 and Figure 5 depict encoding examples of the Length octet(s).

8	7	6	5	4	3	2	1
(msb)	value of the length octet as defined above						(lsb)

Figure 4 – Length octet (one-octet format)

first octet	15	second and third octets	1
11111111	(msb)	value of the length octets as defined above	(lsb)

Figure 5 – Length octets (three-octet format)

5.3.4 Contents octet(s)

The Contents octet(s) shall encode the data value according to the encoding rule defined for its type.

The Contents octet(s) shall have either of the following two forms: primitive encoding or constructed encoding.

a) If the Contents octet(s) contain a primitive encoding, they represent an encoding of one value.

b) If the Contents octet(s) contain a constructed encoding, they represent an enumerated encoding of more than one value.

5.4 Data type encoding rules

5.4.1 General

5.4.1.1 Boolean

A Boolean value shall be encoded as follows.

a) The Identifier octet and the Length octet(s) shall not be present.

b) The Contents octet(s) component always consists of one octet. If the Boolean value equals FALSE, all bits of the octet are 0. If the Boolean value equals TRUE, the octet can contain any combination of bits other than the encoding for FALSE.

5.4.1.2 Integer

An Integer value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present if the size of the Integer value is invariable. An integer with invariable size is created by constraining the possible value. The Length octet(s) shall be present if the size of the Integer value is variable.

c) The Contents octet(s) shall contain the two's complement binary number equal to the Integer value. The most significant eight bits of the Integer value are encoded in bit 8 to bit 1 of the first octet, the next eight bits in bit 8 to bit 1 of the next octet and so on. If the values of an Integer type are restricted to negative and non-negative numbers, bit 8 of the

first octet gives the sign of the value if the values are restricted to non-negative numbers only, no sign bit is needed.

5.4.1.3 Unsigned value

An Unsigned Value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present if the size of the Unsigned Value is invariable. The length of an Unsigned Value with invariable depends on the specified range of the value. The Length octet(s) shall be present if the size of the Unsigned Value is variable.

c) The Contents octet(s) shall be a binary number equal to the Unsigned Value, and consist of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn, up to and including the last octet of the Contents octet(s).

5.4.1.4 Floating Point

A Floating Point value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present.

c) The Contents octet(s) shall contain floating point values defined in conformance with the IEC 60559. The sign is encoded by using bit 8 of the first octet. It is followed by the exponent starting from bit 7 of the first octet, and then the mantissa starting from bit 7 of the second octet for Floating32 and from bit 4 of the second octet for Floating64.

5.4.1.5 Bit string

A Bit String value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present if the size of the Bit String value is invariable. A Bit String with invariable size is created by applying a size constraint containing only one value on the Bit String type. The Length octet(s) shall be present if the size of the Bit String value is variable.

c) The Contents octet(s) comprise as many octets as necessary to contain all bits of the actual value: N_octets = (N_Bits-1) div 8 + 1. The Bit String value commencing with the first bit and proceeding to the trailing bit shall be placed in bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet and so on. If the number of bits is not a multiple of 8, there are so-called unused bits, which are located in the least significant bits of the last octet. The value of the unused bits may be zero (0) or one (1) and carry no meaning.

5.4.1.6 Octet string

An octet String value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present if the size of the octet String value is invariable. An octet String with invariable size is created by applying a size constraint containing only one value on the octet String type. The Length octet(s) shall be present if the size of the octet String value is variable.

c) The Contents octet(s) shall be equal in value to the octets in the data value.

5.4.1.7 Visible string

A Visible String value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present if the size of the Visible String value is invariable. A Visible String with invariable size is created by applying a size constraint containing

only one value on the Visible String type. The Length octet(s) shall be present if the size of the Visible String value is variable.

c) The Contents octet(s) shall be equal in value to the octets in the data value.

5.4.1.8 UNICODE string (ISO 10646 string)

A UNICODE String value shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall indicate the number of octets in the Contents octet(s) as a binary number.

c) Each ISO 10646 character shall be placed in two octets in the Contents octet(s), with the high-order octet placed in the first octet and the low-order octet in the subsequent octet, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents octet(s).

5.4.1.9 Binary time

A Binary Time value shall be encoded as follows.

a) The Identifier octet shall not be present

b) The Length octet(s) shall not be present.

c) The Contents octet(s) shall be a binary number equal to the Binary Time value and consisting of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn, up to and including the last octet of the Contents octet(s).

5.4.1.10 BCD type

a) A BCD value shall be encoded as an Unsigned8 value.

b) A BCD value shall be placed in bits 4 to 1 of the Contents octet of an Unsigned8 value. The values of the bits 8 to 5 shall be zero (0).

5.4.1.11 Compact Boolean array

A Compact Boolean Array value shall be encoded as a Bit String value.

5.4.1.12 Compact BCD array type

a) A Compact BCD Array value shall be encoded as a primitive type.

b) The Identifier octet shall not be present.

c) The Length octet(s) shall indicate the number of octets in the array as a binary number.

d) If the number of BCD values is zero, there shall be no subsequent octets, and the Length octet(s) shall be zero.

e) The first BCD value shall be placed as a binary number in bits 8 to 5 of the first Contents octet(s), and the second BCD value shall be placed in bits 4 to 1 of the first Contents octet(s). This will be repeated for the remaining BCD values and Contents octet(s) up to and including the last octet of the Contents octet(s). The values of any unused bits in the last Contents octet shall be set to 1.

5.4.1.13 SEQUENCE type

A value of a SEQUENCE type shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall be present and specify the number of the first level components of the Contents octet(s). However, for the first Keyword “SEQUENCE” of FalArPDU, this length shall not be encoded.

c) The Contents octet(s) shall consist of the encodings of all the element types in the same order as they are specified in the ASN.1 description of the SEQUENCE type.

5.4.1.14 SEQUENCE OF type

A value of a SEQUENCE OF type shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall be present and specify the number of the first-level components of the Contents octet(s).

c) The Contents octet(s) shall consist of the encodings of all the element types in the same order as they are specified in the ASN.1 description of the SEQUENCE OF type.

5.4.1.15 CHOICE type

A value of a CHOICE type shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present.

c) The Contents octet(s) shall consist of the encoding of the selected type of the alternative type list.

5.4.1.16 Null

A value of a NULL type shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present.

c) The Contents octet(s) shall not be present.

5.4.1.17 Tagged type

A value of a Tagged type shall be encoded as follows.

a) The Identifier octet shall only be present if the tagged type is a part of an alternative type list in a CHOICE construct.

b) The Length octet(s) shall not be present.

c) The Contents octet(s) shall consist of the encoding of the type that was tagged.

5.4.1.18 IMPLICIT type

A value of an IMPLICIT type shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present.

c) The Contents octet(s) shall consist of the encoding of the type being referenced by the IMPLICIT construct, except for the case when the referenced type is a SEQUENCE type. In this case, the Contents octet(s) consist only of the Contents octet(s) of the referenced SEQUENCE type, and the Length octet(s) of this SEQUENCE type shall not be present.

5.4.1.19 OPTIONAL and DEFAULT types

A value of an OPTIONAL or DEFAULT type shall be encoded as follows.

a) The Identifier octet shall not be present.

b) The Length octet(s) shall be present. If there is no value for this type, the Length octet(s) contain the value 0.

c) The Contents octet(s) shall consist of the encoding of the referenced type if there is a value for this type, otherwise no Contents octets exist.

5.4.1.20 ANY type

An ANY type is used for the definition of complex types, whose structure is described informally rather than in ASN.1.

A value of an ANY type shall be encoded as follows:

a) The Identifier octet shall not be present.

b) The Length octet(s) shall not be present.

c) The Contents octets shall consist of the encoding of all implicit types that constitute the ANY type.

6 FAL protocol state machines structure

This subclause specifies protocol machines of the FAL and the Interface between them.

NOTE The state machines specified in this clause and ARPMs defined in the following clauses only define the protocol-related events for each. It is a local matter to handle other events.

The behaviour of the FAL is described by three integrated protocol machines. The three kinds of protocol machines are: FAL Service Protocol Machines (FSPMs), the Application Relationship Protocol Machines (ARPMs), and the data-link layer Mapping Protocol Machines (DMPMs). Specific protocol machines are defined for different AREP types. The relationships among these protocol machines as well as primitives exchanged among them are depicted in Figure 6.

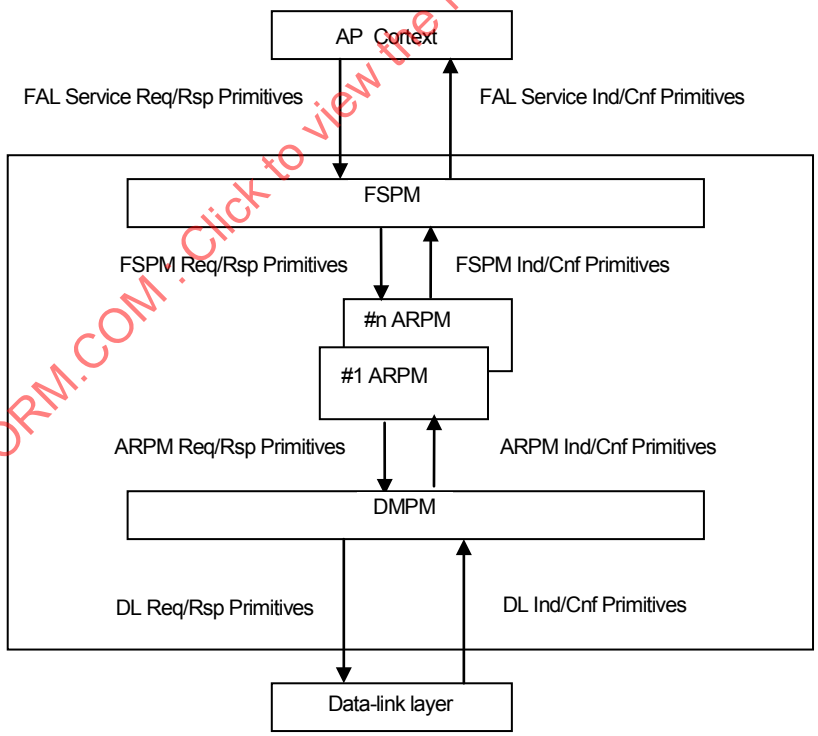


Figure 6 – Relationships among protocol machines and adjacent layers

The FSPM is responsible for the following activities:

a) to accept service primitives from the FAL service user and convert them into FAL internal primitives;

b) to select an appropriate ARPM state machine based on the AREP Identifier parameter supplied by the AP-Context and send FAL internal primitives to the selected ARPM;

c) to accept FAL internal primitives from the ARPM and convert them into service primitives for the AP-Context;

d) to deliver the FAL service primitives to the AP-Context based on the AREP Identifier parameter associated with the primitives.

The ARPM is responsible for the following activities:

a) to accept FAL internal primitives from the FSPM and create and send other FAL internal primitives to either the FSPM or the DMPM, based on the AREP and primitive types;

b) to accept FAL internal primitives from the DMPM and send them to the FSPM in a converted form for the FSPM;

c) if the primitives are for the Establish or Abort service, it shall try to establish or release the specified AR.

The DMPM describes the mapping between the FAL and the DLL. It is common to all the AREP types and does not have any state changes. The DMPM is responsible for the following activities:

a) to accept FAL internal primitives from the ARPM, prepare DLL service primitives, and send them to the DLL;

b) to receive DLL indication or confirmation primitives from the DLL and send them to the ARPM in a converted form for the ARPM.

7 AP-context state machine

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

8 FAL service protocol machines (FSPMs)

8.1 General

There are FAL Service Protocol Machines as follows:

- Variable ASE Protocol Machine (VARM)
- Event ASE Protocol Machine (EVTM)
- Load Region ASE Protocol Machine (LDRM)
- Function Invocation ASE Protocol Machine (FNIM)
- Time ASE Protocol Machine (TIMM)
- Network Management ASE Protocol Machine (NWMM)

8.2 Common parameters of the primitives

Many services have the following parameters. Instead of defining them with each service, the following common definitions are provided.

AREP
This parameter contains sufficient information to identify the AREP to be used to convey the service. This parameter may use a key attribute of the AREP to identify the application relationship. When an AREP supports multiple contexts (established using the Initiate service) at the same time, the AREP parameter is extended to identify the context as well as the AREP.

InvokeID
This parameter identifies this invocation of the service. It is used to associate a service request with its response. Therefore, no two outstanding service invocations can be identified by the same InvokeID value.

Error Info
This parameter provides error information for service errors. It is returned in confirmed service primitives and response primitives.

8.3 Variable ASE protocol machine (VARM)

8.3.1 Primitive definitions

8.3.1.1 Primitives exchanged

Table 3 shows the service primitives, including their associated parameters exchanged between the FAL user and the VARM.

Table 3 – Primitives exchanged between FAL user and VARM

Primitive name	Source	Associated parameters	Functions
Read.req	FAL User	VariableSpecifier	This primitive is used to read values from remote variables.
Write.req	FAL User	VariableSpecifier	This primitive is used to write values to remote variables.
InfReport.req	FAL User	VariableSpecifier, Value, RemoteArep	This primitive is used to publish variables.
Read.rsp	FAL User	VariableSpecifier, Value, ErrorInfo	This primitive is used to convey values of variables requested.
Write.rsp	FAL User	VariableSpecifier, ErrorInfo	This primitive is used to report result of writing requested.
Read.ind	VARM	VariableSpecifier	This primitive is used to convey a read request.
Write.ind	VARM	VariableSpecifier Value	This primitive is used to convey a write request.
InfReport.ind	VARM	VariableSpecifier, Value	This primitive is used to report values of variables published.
Read.cnf	VARM	VariableSpecifier, Value ErrorInfo	This primitive is used to convey values of variables requested and result of reading.
Write.cnf	VARM	VariableSpecifier, ErrorInfo	This primitive is used to report result of writing requested.

8.3.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the VARM are listed in Table 4.

Table 4 – Parameters used with primitives exchanged FAL user and VARM

Parameter name	Description
VariableSpecifier	This parameter specifies a variable or a variable list.
RemoteArep	This parameter specifies a remote AREP to which APDU is to be transferred.
Value	This parameter contains the value of variable to be read/write.
ErrorInfo	This parameter provides error information for service errors.

8.3.2 State machine

8.3.2.1 General

The VARM State Machine has only one possible state: ACTIVE.

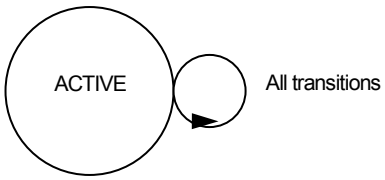


Figure 7 – State transition diagram of VARM

8.3.2.2 State tables

The VARM state machine is described in Figure 7, and in Table 5 and Table 6.

Table 5 – VARM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	Read.req
		=>
		ArepID := GetArep(VariableSpecifier)
		SelectArep(ArepID, “PTC-AR”),
		CS_req{
		user_data := Read-RequestPDU
		}
			ACTIVE

S2	ACTIVE	Write.req
		=>
		ArepID := GetArep(VariableSpecifier)
		SelectArep(ArepID, “PTC-AR”),
		CS_req{
		user_data := Write-RequestPDU
		}
			ACTIVE

S3	ACTIVE	InfReport.req
		=>
		SelectArep(RemoteArep, “MSU-AR”),
		UCS_req{
		user_data := InformationReport-RequestPDU
		}
			ACTIVE

S4	ACTIVE	Read.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Read-ResponsePDU
		}
			ACTIVE

S5	ACTIVE	Write.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Write-ResponsePDU
		}
			ACTIVE

Table 6 – VARM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	ACTIVE	CS_ind
		&& PDU_Type = Read_RequestPDU
		=>
		Read.ind{
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	CS_ind
		&& PDU_Type = Write_RequestPDU
		=>
		Write.ind{
		ArepID := arep_id
		Data := user_data,
		}
			ACTIVE

R3	ACTIVE	CS_ind
		&& PDU_Type = Read_ResponsePDU
		&& GetErrorInfo() = “success”
		=>
		Read.cnf(+){
		Data := user_data
		}
			ACTIVE

R4	ACTIVE	CS_ind
		&& PDU_Type = Read_ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Read.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R5	ACTIVE	CS_ind
		&& PDU_Type =Write_ResponsePDU
		&& GetErrorInfo() = “success”
		=>
		Write.cnf(+){
		Data := user_data
		}
			ACTIVE

R6	ACTIVE	CS_ind
		&& PDU_Type = Write_ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Write.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R7	ACTIVE	UCS_ind
		&& PDU_Type = InformationReport-RequestPDU
		=>
		InfReport.ind{
		Data := user_data
		}
			ACTIVE

R8	ACTIVE	CS_cnf && Status = “success” => (no actions taken)	ACTIVE

R9	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Read”
		=>
		Read.cnf(-){
		ErrorInfo := Status
		}
			ACTIVE

R10	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Write”
		=>
		Write.cnf(-){
		ErrorInfo := Status
		}
			ACTIVE

8.3.2.3 Functions

Table 7 lists the functions used by the VARM, their arguments and their descriptions.

Table 7 – Functions used by the VARM

Function name	Parameter	Description
SelectArep	ArepID, ARtype	Looks for the AREP entry that is specified by the ArepID and AR type
GetArep	VariableSpecifier	Look for the ArepID based on the specified VariableSpecifier.
GetErrorInfo		Gets error information from the APDU
GetService	InvokeID	Gets service name from the InvokeID

8.4 Event ASE protocol machine (EVTM)

8.4.1 Primitive definitions

8.4.1.1 Primitives exchanged

Table 8 shows the service primitives, including their associated parameters exchanged between the FAL user and the EVTM.

Table 8 – Primitives exchanged between FAL user and EVTM

Primitive name	Source	Associated parameters	Functions
Notification.req	FAL User	AREP NotifierID Sequence Number ListOfEventMessages	This primitive is used to request publishing of event messages
EventRecovery.req	FAL User	AREP NotifierID SequenceNumber	This primitive is used to request retransmission of event notification
Notification.ind	EVTM	AREP NotifierID SequenceNumber List of Event Messages	This primitive is used to inform event notification.
EventRecovery.ind	EVTM	AREP NotifierID SequenceNumber	This primitive is used to inform request of retransmission of event notification

8.4.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the EVTM are listed in Table 9.

Table 9 – Parameters used with primitives exchanged FAL user and EVTM

Parameter name	Description
NotifierID	This conditional parameter identifies the notifier issuing the event notification. It is present if the AP has more than one notifier defined for it
SequenceNumber	This optional parameter is the sequence number for the event notification. It may be used for notification recovery purposes
NotificationTime	This optional parameter is the time of the event notification
ListOfEventMessages	This parameter contains the list of event messages that are to be reported. It may contain messages from one or more event objects, and each object contains the same set of parameters

8.4.2 State machine

8.4.2.1 General

The EVTM State Machine has only one possible state: ACTIVE.

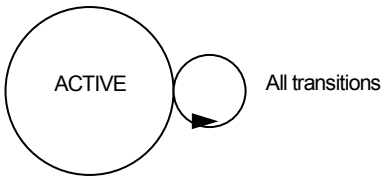


Figure 8 – State transition diagram of EVTM

8.4.2.2 State tables

The EVTM state machine is described in Figure 8, and in Table 10 and Table 11.

Table 10 – EVTM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	Notification.req
		=>
		SelectArep(RemoteArep, “MTU-AR”),
		UCS_req{
		user_data := Event-NotificationPDU
		}
			ACTIVE

S2	ACTIVE	EventRecovery.req
		=>
		SelectArep(RemoteArep, “PTU-AR”),
		UCS_req{
		arep := SelectArep(CalledAREP, “PTU-AR”),
		user_data := EventRecovery-RequestPDU
		}
			ACTIVE

Table 11 – EVTM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	ACTIVE	UCS_ind
		&& PDU_Type = Event_NotifiationPDU
		=>
		Notification.ind{
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	UCS_ind
		&& PDU_Type = EventRecovery-RequestPDU
		=>
		EventRecovery.ind {
		Data := user_data
		}
			ACTIVE

8.4.2.3 Functions

Table 12 lists the function used by the EVTM, their arguments, and their description.

Table 12 – Functions used by the EVTM

Function name	Parameter	Description
SelectArep	ArepID, ARtype	Looks for the AREP entry that is specified by the ArepID and AR type

8.5 Load region ASE protocol machine (LDRM)

8.5.1 Primitive definitions

8.5.1.1 Primitives exchanged

Table 13 shows the service primitives, including their associated parameters exchanged between the FAL user and the LDRM.

Table 13 – Primitives exchanged between FAL user and LDRM

Primitive name	Source	Associated parameters	Functions
Download.req	FAL User	AREP InvokeID LoadRegion LoadData	This primitive is used to request download data to the region
Upload.req	FAL User	AREP InvokeID LoadRegion	This primitive is used to request upload data from the region
Download.rsp	FAL User	AREP InvokeID Error Info	This primitive is used to report result of download requested
Upload.rsp	FAL User	AREP InvokeID LoadData ErrorInfo	This primitive is used to convey data to be uploaded
Download.ind	LDRM	AREP InvokeID LoadRegion LoadData	This primitive is used to convey data downloaded
Upload.ind	LDRM	AREP InvokeID Load region	This primitive is used to convey an upload request
Download.cnf	LDRM	AREP InvokeID ErrorInfo	This primitive is used to convey a result of download
Upload.cnf	LDRM	AREP InvokeID LoadData ErrorInfo	This primitive is used to convey data uploaded

8.5.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the LDRM are listed in Table 14.

Table 14 – Parameters used with primitives exchanged FAL user and LDRM

Parameter name	Description
LoadRegion	This parameter specifies the region from/to which the image is to be loaded
LoadData	This parameter contains the data to be loaded
ErrorInfo	This parameter provides error information for service errors

8.5.2 State machine

8.5.2.1 General

The LDRM State Machine has only one possible state: ACTIVE.

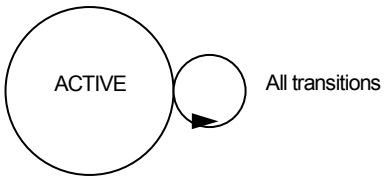


Figure 9 – State transition diagram of LDRM

8.5.2.2 State tables

The LDRM state machine is described in Figure 9, and in Table 15 and Table 16.

Table 15 – LDRM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	Download.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := DownLoad-RequestPDU
		}
			ACTIVE

S2	ACTIVE	Upload.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := UpLoad-RequestPDU
		}
			ACTIVE

S3	ACTIVE	Download.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		arep := SelectArep(CallingAREP, “PTC-AR”),
		user_data := DownLoad-ResponsePDU
		}
			ACTIVE

S4	ACTIVE	Upload.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		arep := SelectArep(CallingAREP, “PTC-AR”),
		user_data := UpLoad-ResponsePDU
		}
			ACTIVE

Table 16 – LDRM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	ACTIVE	CS_ind
		&& PDU_Type = DownLoad-RequestPDU
		=>
		Download.ind {
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	CS_ind
		&& PDU_Type = UpLoad-RequestPDU
		=>
		Upload.ind {
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R3	ACTIVE	CS_ind
		&& PDU_Type = DownLoad-ResponsePDU
		=>
		Download.cnf(+) {
		Data := user_data
		}
			ACTIVE

R4	ACTIVE	CS_ind
		&& PDU_Type = UpLoad-ResponsePDU
		=>
		Upload.cnf(+) {
		Data := user_data
		}
			ACTIVE

R5	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Download”
		=>
		Download.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

R6	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Upload”
		=>
		Upload.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

8.5.2.3 Functions

Table 17 lists the functions used by the LDRM, their arguments, and their descriptions.

Table 17 – Functions used by the LDRM

Function name	Parameter	Description
SelectArep	ArepID, ARtype	Looks for the AREP entry that is specified by the ArepID and AR type
GetErrorInfo		Gets error information from the APDU.
GetService	InvokeID	Gets service name from the InvokeID.

8.6 Function invocation ASE protocol machine (FNIM)

8.6.1 Primitive definitions

8.6.1.1 Primitives exchanged

Table 18 shows the service primitives, including their associated parameters exchanged between the FAL user and the FNIM.

Table 18 – Primitives exchanged between FAL user and FNIM

Primitive name	Source	Associated parameters	Functions
Start.req	FAL User	AREP InvokeID FunctionID	This primitive is used to request start of the function
Stop.req	FAL User	AREP InvokeID FunctionID	This primitive is used to request stop of the function.
Resume.req	FAL User	AREP InvokeID FunctionID	This primitive is used to request resume of the function.
Start.rsp	FAL User	AREP InvokeID Error Info	This primitive is used to report result of start requested.
Stop.rsp	FAL User	AREP InvokeID Error Info	This primitive is used to report result of stop requested.
Resume.rsp	FAL User	AREP InvokeID Error Info	This primitive is used to report result of resume requested.
Start.ind	FNIM	AREP InvokeID FunctionID	This primitive is used to convey a start request.
Stop.ind	FNIM	AREP InvokeID FunctionID	This primitive is used to convey a stop request.
Resume.ind	FNIM	AREP InvokeID FunctionID	This primitive is used to convey a resume request.
Start.cnf	FNIM	AREP InvokeID Error Info	This primitive is used to convey a result of start.
Stop.cnf	FNIM	AREP InvokeID Error Info	This primitive is used to convey a result of stop.
Resume.cnf	FNIM	AREP InvokeID Error Info	This primitive is used to convey a result of resume.

8.6.1.2 Parameters of primitives

The parameter used with the primitives exchanged between the FAL user and the FNIM is listed in Table 19.

Table 19 – Parameters used with primitives exchanged FAL user and FNIM

Parameter name	Description
FunctionID	This parameter specifies one of the key attributes of the function invocation object

8.6.2 State machine

8.6.2.1 General

The FNIM State Machine has only one possible state: ACTIVE.

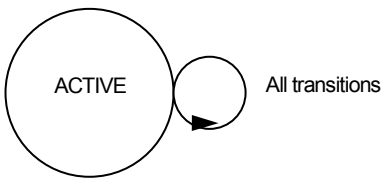


Figure 10 – State transition diagram of FNIM

8.6.2.2 State tables

The FNIM state machine is described in Figure 10, and in Table 20 and Table 21.

Table 20 – FNIM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	Start.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := Start-RequestPDU
		}
			ACTIVE

S2	ACTIVE	Stop.req.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := Stop-RequestPDU
		}
			ACTIVE

S3	ACTIVE	Resume.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := Resume-ResponsePDU
		}
			ACTIVE

S4	ACTIVE	Start.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Start-ResponsePDU
		}
			ACTIVE

S5	ACTIVE	Stop.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Stop-ResponsePDU
		}
			ACTIVE

S6	ACTIVE	Resume.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Resume-ResponsePDU
		}
			ACTIVE

Table 21 – FNIM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	ACTIVE	CS_ind
		&& PDU_Type = Start-RequestPDU
		=>
		Start.ind {
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	CS_ind
		&& PDU_Type = Stop-RequestPDU
		=>
		Stop.ind {
		Data := user_data
		}
			ACTIVE

R3	ACTIVE	CS_ind
		&& PDU_Type = Resume-RequestPDU
		=>
		Resume.ind {
		Data := user_data
		}
			ACTIVE

#	Current state	Event or condition => action	Next state

R4	ACTIVE	CS_ind
		&& PDU_Type = Start-ResponsePDU
		=>
		Start.cnf(+) {
		Data := user_data
		}
			ACTIVE

R5	ACTIVE	CS_ind
		&& PDU_Type = Start-ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Start.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R6	ACTIVE	CS_ind
		&& PDU_Type = Stop-ResponsePDU
		=>
		Stop.cnf(+) {
		Data := user_data
		}
			ACTIVE

R7	ACTIVE	CS_ind
		&& PDU_Type = Stop-ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Stop.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R8	ACTIVE	CS_ind
		&& PDU_Type = Resume-ResponsePDU
		=>
		Resume.cnf(+) {
		Data := user_data
		}
			ACTIVE

R9	ACTIVE	CS_ind
		&& PDU_Type = Resume-ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Resume.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R10	ACTIVE	CS_cnf && Status = “success” => (no actions taken)	ACTIVE

R11	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Start”
		=>
		Start.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

R12	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Stop”
		=>
		Stop.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

R13	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Resume”
		=>
		Resume.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

8.6.2.3 Functions

Table 22 lists the functions used by the FNIM, their arguments, and their descriptions.

Table 22 – Functions used by the FNIM

Function name	Parameter	Description
SelectArep	AREPid, ARtype	Looks for the AREP entry that is specified by the AREPid and AR type
GetErrorInfo		Gets error information from the APDU
GetService	InvokeID	Gets service name from the InvokeID

8.7 Time ASE protocol machine (TIMM)

8.7.1 Primitive definitions

8.7.1.1 Primitives exchanged

Table 23 shows the service primitives, including their associated parameters exchanged between the FAL user and the TIMM.

Table 23 – Primitives exchanged between FAL user and TIMM

Primitive name	Source	Associated parameters	Functions
GetTime.req	FAL User	AREP InvokeID	This primitive is used to request network time
SetTim.req	FAL User	AREP InvokeID NetworkTime	This primitive is used to request setting of time to the network.
SetTim.ind	TIMM	AREP InvokeID Network-time	This primitive is used to report setting of network time.
Tick.ind	TIMM	Tick	This primitive is used to report periodical trigger synchronized to network time.
GetTim.cnf	TIMM	AREP InvokeID NetworkTime ErrorInfo	This primitive is used to convey a result of getting of network time.
SetTim.cnf	TIMM	AREP InvokeID ErrorInfo	This primitive is used to convey a result of setting of network time.

8.7.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the TIMM are listed in Table 24.

Table 24 – Parameters used with primitives exchanged FAL user and TIMM

Parameter name	Description
NetworkTime	This parameter is the value of the network time
ErrorInfo	This parameter provides error information for service errors.
Tick	This parameter indicates tick timing.

8.7.2 State machine

8.7.2.1 General

The TIMM State Machine has four possible states. The defined states and their descriptions are shown in Table 25 and Figure 11.

Table 25 – TIMM states

State	Description
TIM_MST	TIMM is acting as network time master
DOM_MST	TIMM is acting as domain time master.
SLAVE	TIMM is synchronized with domain time master.
IDLE	TIMM is not synchronized with network time.

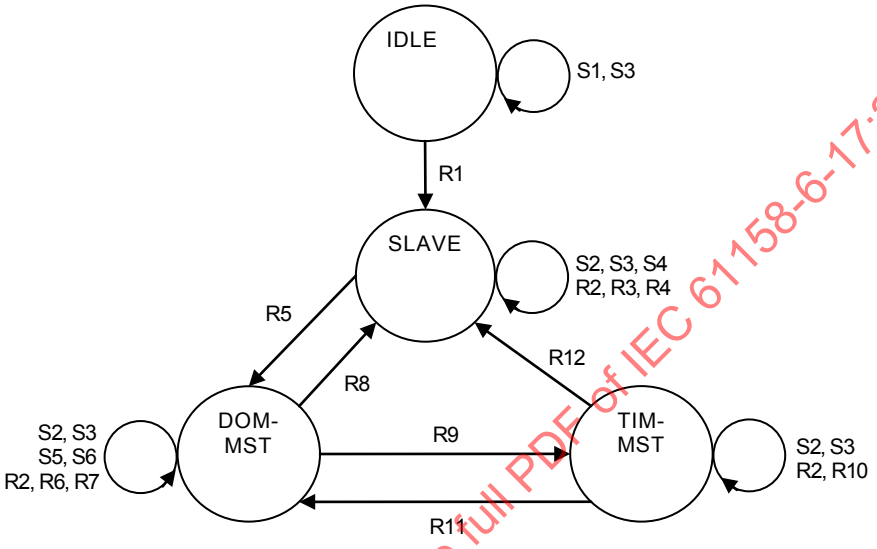


Figure 11 – State transition diagram of TIMM

8.7.2.2 State tables

The TIMM state machine is described in Figure 11, and in Table 26 and Table 27.

Table 26 – TIMM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	IDLE	GetTime.req
		=>
		GetTime.cnf {
		Error info := “not synchronized”
		}
			IDLE

S2	SLAVE
	DOM_MST
	TIM_MST
		GetTime.req
		=>
		GetTime.cnf {
		NetworkTime := GetLocalTime()
		}
			SAME

S3	ANY	SetTim.req
		=>
		SelectArep(“NET”, “MTU-AR”),
		UCS_req {
		user_data := SetTime-RequestPDU
		}
			SAME

S4	SLAVE	CheckTimer(Timer1) = “Expired” => SelectArep (“DOM-MST”, “PTC-AR”), CS_req { user_data := Time-RequestPDU }, StartTimer(Timer1)	SLAVE
S5	DOM-MST	CheckTimer(Timer2) = “Expired” => SelectArep(“DOM”, “MTU-AR”), UCS_req { user_data := TimeDistribute-RequestPDU }, StartTimer(Timer2)	DOM-MST
S6	DOM-MST	CheckTimer(Timer3) = “Expired” => SelectArep(“TIM-MST”, “PTC-AR”), CS_req { user_data := Time-RequestPDU }, StartTimer(Timer3)	DOM-MST

Table 27 – TIMM state table – Receiver transitions

#	Current state	Event or condition => action	Next state
R1	IDLE	UCS_ind && PDU_Type = TimeDistribute-RequestPDU =>	SLAVE
R2	SLAVE DOM_MST TIM_MST	CheckTimer(Tick) = “Expired” => Tick.ind {} StartTimer(Tick)	SAME
R3	SLAVE	UCS_ind && PDU_Type = TimeDistribute-RequestPDU => UpdateLocalTime(DelayFactor)	SLAVE
R4	SLAVE	CS_ind && PDU_Type = Time-ResponsePDU => DelayFactor = CalcurateDelay()	SLAVE
R5	SLAVE	CheckNW() == DOM-MST => (no actions taken)	DOM-MST

R6	DOM-MST	CS_ind
		&& PDU_Type = Time-RequestPDU
		=>
		SelectArep(CallingAREP, “PTC-AR”),
		CS_rsp {
		user_data := Time-ResponsePDU
		}
			DOM-MST

R7	DOM-MST	CS_ind && PDU_Type = Time-ResponsePDU => DelayFactor = CalcurateDelay(), UpdateLocalTime(DelayFactor)	DOM-MST
R8	DOM-MST	CheckNW() == SLAVE => (no actions taken)	SLAVE
R9	DOM-MST	CheckNW() == TIM-MST => (no actions taken)	TIM-MST

R10	TIM-MST	CS_ind
		&& PDU_Type = Time-RequestPDU
		=>
		SelectArep(CallingAREP, “PTC-AR”),
		CS_rsp {
		user_data := Time-ResponsePDU
		}
			TIM-MST

R11	TIM-MST	CheckNW() == DOM-MST => (no actions taken)	DOM-MST
R12	TIM-MST	CheckNW() == SLAVE => (no actions taken)	SLAVE

8.7.2.3 Functions

Table 28 lists the functions used by the TIMM, their arguments, and their descriptions.

Table 28 – Functions used by the TIMM

Function name	Parameter	Description
SelectArep	ArepID, ARtype	Looks for the AREP entry that is specified by the ArepID and AR type. The value “DOM” for ArepID specifies all stations of the domain to which the NWMM belong. The value “NET” for ArepID specifies all stations of the network. The value “DOM-MST” for ArepID specifies the AREP of the domain time master of the domain to which the NWMM belong. The value “TIM-MST” for ArepID specifies the AREP of the network time master
GetLocalTime		Gets local time from the internal clock
UpdateLocalTime	DelayFactor	Updates local clock with the received time and the delay factor
CalcurateDelay		Calculate the delay factor from received APDU
CheckTimer	TimerID	Checks status of the specified timer. If the timer has been expired, the value “Expired” is returned
StartTimer	TimerID	Starts the timer specified

8.8 Network management ASE protocol machine (NWMM)

8.8.1 Primitive definitions

8.8.1.1 Primitives exchanged

Table 29 shows the service primitives, including their associated parameters exchanged between the FAL user and the NWMM.

Table 29 – Primitives exchanged between FAL user and NWMM

Primitive name	Source	Associated parameters	Functions
GetNW.req	FAL User	InvokeID	This primitive is used to request network status
GetSTN.req	FAL User	InvokeID StationID	This primitive is used to request station status.
NWStatus.ind	NWMM	NetworkStatus	This primitive is used to report changes of network status.
STNStats.ind	NWMM	StationID StationStatus RouteStatus	This primitive is used to report changes of station status.
GetNW.cnf	NWMM	InvokeID NetworkStatus	This primitive is used to convey network status.
GetSTN.cnf	NWMM	InvokeID StationStatus RouteStatus	This primitive is used to convey station status requested.

8.8.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the NWMM are listed in Table 30.

Table 30 – Parameters used with primitives exchanged FAL user and NWMM

Parameter name	Description
StationID	This parameter indicates a station
StationStatus	This parameter indicates status of station which is specified in the request primitive.
RouteStatus	This parameter indicates status of routes for the station which is specified in the request primitive.
NetworkStatus	This parameter indicates consistency of the primary network and the secondary network.

8.8.2 State machine

8.8.2.1 General

The NWMM State Machine has three possible states. The defined states and their descriptions are shown in Table 31 and Figure 12.

Table 31 – NWMM states

State	Description
MST	NWMM as a domain master.
SLAVE	NWMM as a slave.

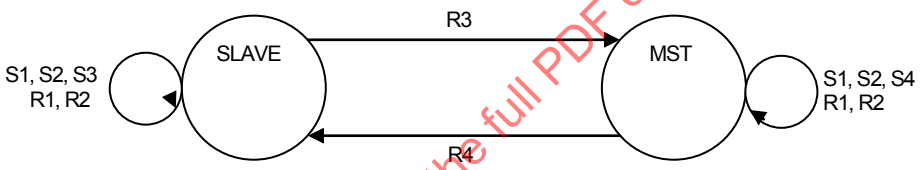


Figure 12 – State transition diagram of NWMM

8.8.2.2 State tables

The NWMM state machine is described in Figure 12, and in Table 32 and Table 33.

Table 32 – NWMM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ANY	GetNW.req
		=>
		GetNW.cnf{
		NetworkStatus := GetNWstatus()
		}
			SAME

S2	ANY	GetSTN.req
		=>
		GetSTN.cnf{
		StationStatus := GetSTNstatus(StationID)
		RouteStatus := GetRoutestatus(StationID)
		}
			SAME

S3	SLAVE	CheckTimer(DiagTimer) => SelectArep("DOM", "MTU-AR"), UCS_req{ user_data := InDiag-RequestPDU }, StartTimer(DiagTimer)	SLAVE

S4	MST	CheckTimer(DiagTimer)
		=>
		SelectArep("DOM", "MTU-AR"),
		UCS_req{
		user_data := InDiag-RequestPDU
		}
		SelectArep("NET", "MTU-AR")
		UCS_req{
		user_data := ExDiag-Request PDU
		},
		StartTimer(DiagTimer)
			MST

Table 33 – NWMM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	ANY	UCS_ind
		&& (PDU_Type = InDiag-RequestPDU
		|| PDU_Type = ExDiag-RequestPDU)
		
		=>
		UpdateNWstatus()
		
		CheckNWstatus(NWstatus-table) = “True”
		=>
		NWStatus.ind{
		NetworkStatus := GetNWstatus()
		}
		
		(changedSTN := CheckSTNstatus(NWstatus-table)) <> “None”
		=>
		STNStats.ind {
		StationID := changed-station,
		StationStatus := GetSTNstatus(StationID)
		RouteStatus := GetRouteStatus(StationID)
		}
			SAME

R2	ANY	CheckTimer(AgingTimer) = “Expired” => UpdateNWstatus() (changedSTN := CheckSTNstatus(NWstatus-table)) <> “None” => STNStats.ind { StationID := changed-station, StationStatus := GetSTNstatus(StationID) RouteStatus := GetRouteStatus(StationID) }, StartTimer(AgingTimer)	SAME
R3	SLAVE	CheckMaster(NWstatus-table) = “True” => (no actions taken)	MST
R4	MST	CheckMaster (NWstatus-table) = “False” => (no actions taken)	SLAVE

8.8.2.3 Functions

Table 34 lists the functions used by the NWMM, their arguments, and their descriptions.

Table 34 – Functions used by the NWMM

Function name	Parameter	Description
GetNWstatus	NWstatus-table	Gets network status from the network status table.
GetSTNstatus	NWstatus-table,StationID	Gets station status of the specified station from the network status table.
GetRoutestatus	NWstatus-table, StationID	Gets route status to the specified station from the network status table.
SelectArep	ArepID, ARtype	Looks for the AREP entry that is specified by the ArepID and AR type. The value “DOM” for ArepID specifies all stations of the domain to which the NWMM belongs. The value “NET” for ArepID specifies all stations of the network.
CheckTimer	TimerID	Checks status of the specified timer. If the timer has expired, the value “Expired” is returned.
StartTimer	TimerID	Starts the specified timer.
UpdateNWstatus	NWstatus-table	Updates the network status table according to received APDU, If aging time of each entry of the network status table has expired, then updates the entry as not valid.
CheckNWstatus()	NWstatus-table	Checks the network status table. If any change of network status is detected, the value “True” is returned.
CheckSTNstatus()	NWstatus-table	Checks the network status table. If any change of station status is detected, the StationID of the detected station is returned.
CheckMaster	NWstatus-table	Checks the network status table. If the NWMM of own station is recognized as master of the domain according to the predefined rules, the value “True” is returned.

9 Application relationship protocol machines (ARPMs)

9.1 General

This fieldbus has Application Relationship Protocol Machines (ARPMs) for

- point-to-point user-triggered confirmed client/server AREP (PTC-AR);
- point-to-point user-triggered unconfirmed client/server AREP (PTU-AR);
- point-to-point network-scheduled unconfirmed client/server AREP (PSU-AR);
- multipoint user-triggered unconfirmed publisher/subscriber AREP (MTU-AR);
- multipoint network-scheduled unconfirmed publisher/subscriber AREP (MSU-AR).

9.2 Primitive definitions

9.2.1 Primitives exchanged

Table 35 lists the primitives, including their associated parameters exchanged between the FSPM and the ARPM.

Table 35 – Primitives exchanged between FSPM and ARPM

Primitive name	Source	Associated parameters	Functions
EST_req	FSPM	Remote_dlsap_address	**This primitive is used to request establishment of the AR**
ABT_req	FSPM	Reason_code	**This primitive is used to request abort of the AR.**
CS_req	FSPM	Destination_dlsap_address, InvokeID, User_data,	**This primitive is used to request sending of the ConfirmedSend-CommandPDU.**
UCS_req	FSPM	Remote_dlsap_address, User_data	**This primitive is used to request sending of the UnconfirmedSend-CommandPDU.**
CS_rsp	FSPM	Source_dlsap_address, User_data	**This primitive is used to request sending of the ConfirmedSend-ResponsePDU.**
CS_ind	ARPM	Source_dlsap_address, InvokeID, User_data	**This primitive is used to report the received ConfirmedSend-CommandPDU.**
UCS_ind	ARPM	Remote_dlsap_address, InvokeID, User_data	**This primitive is used to report the received UnconfirmedSend-CommandPDU.**
EST_cnf	ARPM	InvokeID, Result,	**This primitive is used to convey a result of AR establishment.**
CS_cnf	ARPM	InvokeID, Result,	**This primitive is used to convey a result of confirmed sending**

9.2.2 Parameters of primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are listed in Table 36.

Table 36 – Parameters used with primitives exchanged FSPM user and ARPM

Parameter name	Description
InvokeID	This parameter is locally used and defined by the user to identify the request
Remote_dlsap_address	This parameter contains the destination DLSAP-address in the request and the source DLSAP-address in the indication.
Destination_dlsap_address	This parameter contains the Destination DLSAP-address.
Source_dlsap_address	This parameter contains the Source DLSAP-address.
User_data	This parameter contains the service dependent body for the APDU.
Result	This parameter indicates that the service request succeeded or failed.
Reason_Code	This parameter indicates the reason for the Abort

9.3 State machine

9.3.1 Point-to-point user-triggered confirmed client/server ARPM (PTC-ARPM)

9.3.1.1 General

The PTC-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 37 and Figure 13.

Table 37 – PTC-ARPM states

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs

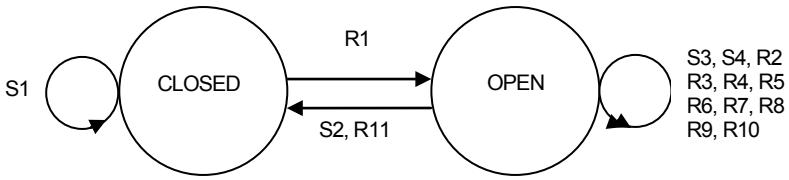


Figure 13 – State transition diagram of the PTC-ARPM

9.3.1.2 States

The PTC-ARPM state machine is described in Figure 13, and in Table 38 and Table 39.

Table 38 – PTC-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	CLOSED	EST_req
		=>
		Establish_req{
		cardinality := “one-to-one”,
		remote_confirm := “True”,
		sequence_control := “True”
		conveyance_policy := “Queue”
		}
			CLOSED

S2	OPEN	ABT_req => Abort_req {} ABT_ind {}	CLOSED

S3	OPEN	CS_req
		&& Role = “Client” || “Peer”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Destination _dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “CS_PDU”,
		fal_data := user_data)
		}
			OPEN

S4	OPEN	CS_rsp
		&& Role = “Server” || “Peer”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Destination _dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “CS_RspPDU”,
		fal_data := user_data)
		}
			OPEN

Table 39 – PTC-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	CLOSED	Establish_cnf
		&& status = “Success”
		=>
		DLSAP_ID := dlsap_id
		EST_cnf {
		Status := status
		}
			OPEN

R2	OPEN	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = “CS_ReqPDU”
		&& Role = “Peer” || “Server”
		=>
		CS_ind{
		Source _dlsap_address := calling_address,
		user _data := fal_pdu
		}
			OPEN

R3	OPEN	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = “CS_RspPDU”
		&& Role = “Client” || “Peer”
		=>
		CS_cnf{
		user_data := fal_pdu
		}
			OPEN

R4	OPEN	FAL-PDU_ind && FAL_Pdu_Type (fal_pdu) <> “CS_ReqPDU” && Role = “Server” => (no actions taken)	OPEN
R5	OPEN	FAL-PDU_ind && FAL_Pdu_Type (fal_pdu) <> “CS_RspPDU” && Role = “Client” => (no actions taken)	OPEN
R6	OPEN	FAL-PDU_ind && FAL_Pdu_Type (fal_pdu) <> “CS_ReqPDU” && FAL_Pdu_Type (fal_pdu) <> “CS_RspPDU” && Role = “Peer” => (no actions taken)	OPEN

R7	OPEN	FAL-PDU_cnf
		&& FALPdu_Type(fal-pdu) = “CS_Req PDU”
		&& Role = “Client” || “Peer”
		&& status <> “success”
		=>
		CS_Cnf {
		user_data := null,
		result := status
		}
			OPEN

R8	OPEN	FAL-PDU_cnf && Role = “Client” || “Peer” && status = “success” => (no actions taken)	OPEN
R9	OPEN	FAL-PDU_Ind && FALPdu_Type(fal-pdu) = “CS_Rsp PDU” && Role = “Server” || “Peer” => (no actions taken)	OPEN
R10	OPEN	ErrorToARPM => (No actions taken. See note.)	OPEN
R11	OPEN	Abort_ind => ABT_ind{}	CLOSED

9.3.2 Point-to-point user-triggered unconfirmed client/server ARPM (PTU-ARPM)

9.3.2.1 General

The PTU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 40 and Figure 14.

Table 40 – PTU-ARPM states

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs

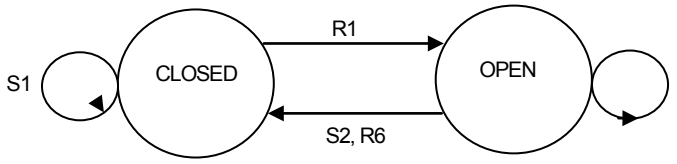


Figure 14 – State transition diagram of the PTU-ARPM

9.3.2.2 State tables

The PTU-ARPM state machine is described in Figure 14, and in Table 41 and Table 42.

Table 41 – PTU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	CLOSED	EST_req
		=>
		Establish_req{
		cardinality := “one-to-one”,
		remote_confirm := “True”,
		sequence_control := “False”
		conveyance_policy := “Queue”
		}
			CLOSED

S2	OPEN	ABT_req => Abort_req {} ABT_ind {}	CLOSED

S3	OPEN	UCS_req
		&& Role = “Client” || “Peer”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Remote_dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “UCS_PDU”,
		fal_data := user_data)
		}
			OPEN

S4	OPEN	UCS_req && Role = “Server” => (no actions taken)	OPEN

Table 42 – PTU-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	CLOSED	Establish_cnf
		&& status = “Success”
		=>
		DLSAP_ID := dlsap_id
		EST_cnf {
		Status := status
		}
			OPEN

R2	OPEN	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = “UCS_PDU”
		&& Role = “Server” || “Peer”
		=>
		CS_ind{
		remote_dlsap_address := calling_address,
		user_data := fal_pdu
		}
			OPEN

R3	OPEN	FAL-PDU_ind && FAL_Pdu_Type (fal_pdu) <> “UCS_PDU” && Role = “Client” || “Peer” => (no actions taken)	OPEN
R4	OPEN	FAL-PDU_ind && Role = “Client” => (no actions taken)	OPEN
R5	OPEN	ErrorToARPM => (No actions taken. See note.)	OPEN
R6	OPEN	Abort_ind => ABT_ind{}	CLOSED

9.3.3 Point-to-point network-scheduled unconfirmed client/server ARMP (PSU-ARPM)

9.3.3.1 General

The PSU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 43 and Figure 15.

Table 43 – PSU-ARPM states

State	Description
CLOSED	The AREP is defined but not capable of sending or receiving FAL-PDUs
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs

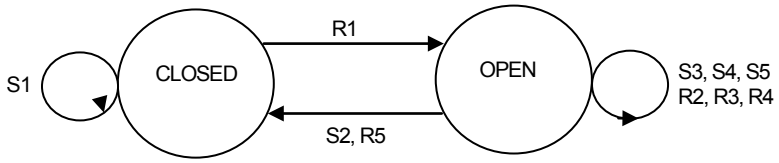


Figure 15 – State transition diagram of the PSU-ARPM

9.3.3.2 State tables

The PSU-ARPM state machine is described in Figure 15, and in Table 44 and Table 45.

Table 44 – PSU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	CLOSED	EST_req
		=>
		Establish_req{
		cardinality := “one-to-one”,
		remote_confirm := “False”,
		sequence_control := “False”
		conveyance_policy := “Buffer”
		}
			CLOSED

S2	OPEN	ABT_req => Abort_req {} ABT_ind {}	CLOSED
S3	OPEN	UCS_req && Role =“PushPublisher” => LoadBuffer(Remote_dlsap_address, user_data)	OPEN
S4	OPEN	StartTransmitCycleTimer expired && Role =“PushPublisher” => FAL-PDU_req { dlsap_id := DLSAP_ID, called_address := Remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := “UCS_PDU” fal_data := local_buf) }, StartTransmitCycleTimer(arep_id)	OPEN
S5	OPEN	UCS_req && Role = “Subscriber” => (no actions taken)	OPEN

Table 45 – PSU-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state
R1	CLOSED	Establish_cnf && status = “Success” => DLSAP_ID := dlsap_id EST_cnf {} StartTransmitCycleTimer(arep_id)	OPEN

R2	OPEN	FAL-PDU_ind
		&& Role = “Subscriber”
		&& FAL_Pdu_Type (fal_pdu) = “UCS_PDU”
		=>
		UCS_ind {
		remote_dlsap_address := calling_address,
		user_data := fal_pdu,
		}
			OPEN

R3	OPEN	FAL-PDU_ind && FAL_Pdu_Type (fal_pdu) <> “UCS_PDU” => (no actions taken)	OPEN
R4	OPEN	FAL-PDU_ind && Role = “Publisher” => (no actions taken)	OPEN
R5	OPEN	Abort_ind => ABT_ind{}	CLOSED

9.3.4 Multipoint user-triggered unconfirmed publisher/subscriber ARPM (MTU-ARPM)

9.3.4.1 General

The MTU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 46 and Figure 16.

Table 46 – MTU-ARPM states

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs

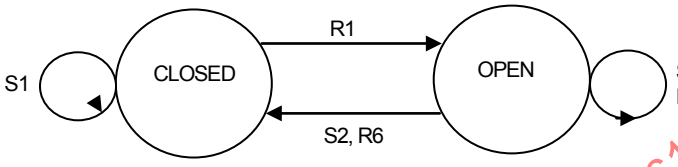


Figure 16 – State transition diagram of the MTU-ARPM

9.3.4.2 State tables

The MTU-ARPM state machine is described in Figure 16, and in Table 47 and Table 48.

Table 47 – MTU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	CLOSED	EST_req
		=>
		Establish_req{
		cardinality := “one-to-many”,
		remote_confirm := “False”,
		sequence_control := “True”
		conveyance_policy := “Queue”
		}
			CLOSED

S2	OPEN	ABT_req => Abort_req {} ABT_ind {}	CLOSED

S3	OPEN	UCS_req
		&& Role = “Publisher”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Remote_dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “UCS_PDU”,
		fal_data := user_data)
		}
			OPEN

S4	OPEN	UCS_req && Role = “Subscriber” => (no actions taken)	OPEN

Table 48 – MTU-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state
R1	CLOSED	Establish_cnf && status = “Success” => DLSAP_ID := dlsap_id EST_cnf {}	OPEN

R2	OPEN	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = “UCS_PDU”
		&& Role = “Subscriber”
		=>
		CS_ind{
		remote_dlsap_address := calling_address,
		user_data := fal_pdu
		}
			OPEN

R3	OPEN	FAL-PDU_ind && FAL_Pdu_Type (fal_pdu) <> “UCS_PDU” => (no actions taken)	OPEN
R4	OPEN	FAL-PDU_ind && Role = “Publisher” => (no actions taken)	OPEN
R5	OPEN	ErrorToARPM => (No actions taken. See note.)	OPEN
R6	OPEN	Abort_ind => ABT_ind{}	CLOSED

9.3.5 Multipoint network-Scheduled Unconfirmed publisher/subscriber ARPM (MSU-ARPM)

9.3.5.1 General

The MSU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 49 and Figure 17.

Table 49 – MSU-ARPM states

State	Description
CLOSED	The AREP is defined but not capable of sending or receiving FAL-PDUs
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs

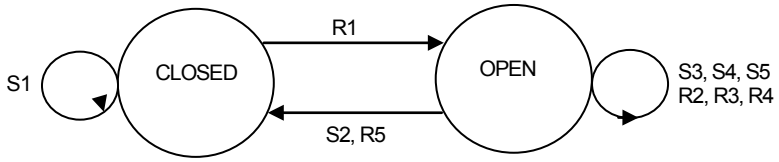


Figure 17 – State transition diagram of the MSU-ARPM

9.3.5.2 State tables

The MSU-ARPM state machine is described in Figure 17, and in Table 50 and Table 51.

Table 50 – MSU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	CLOSED	EST_req
		=>
		Establish_req{
		cardinality := “one-to-many”,
		remote_confirm := “False”,
		sequence_control := “False”
		conveyance_policy := “Buffer”
		}
			CLOSED

S2	OPEN	ABT_req => Abort_req {} ABT_ind {}	CLOSED
S3	OPEN	UCS_req && Role =“Publisher” => LoadBuffer(Remote_dlsap_address, user_data)	OPEN
S4	OPEN	StartTransmitCycleTimer expired && Role =“Publisher” => FAL-PDU_req { dlsap_id := DLSAP_ID, called_address := Remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := “UCS_PDU”, fal_data := local_buf) }, StartTransmitCycleTimer(arep_id)	OPEN
S5	OPEN	UCS_req && Role = “Subscriber” => (no actions taken)	OPEN

Table 51 – MSU-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state
R1	CLOSED	Establish_cnf && status = “Success” => DLSAP_ID := dlsap_id EST_cnf {} StartTransmitCycleTimer(arep_id)	OPEN

R2	OPEN	FAL-PDU_ind
		&& Role = “Subscriber”
		&& FAL_Pdu_Type (fal_pdu) = “UCS_PDU”
		=>
		UCS_ind {
		remote_dlsap_address := calling_address,
		user_data := fal_pdu,
		}
			OPEN

R3	OPEN	FAL-PDU_ind && || FAL_Pdu_Type (fal_pdu) <> “UCS_PDU” => (no actions taken)	OPEN
R4	OPEN	FAL-PDU_ind && Role = “Publisher” => (no actions taken)	OPEN
R5	OPEN	Abort_ind => ABT_ind{}	CLOSED

9.4 Functions

Table 52 lists the functions used by the ARPMs, their arguments, and their descriptions.

Table 52 – Functions used by the ARPMs

Function name	Parameter	Description
BuildFAL-PDU	fal_pdu_name, fal_data	This function builds an FAL-PDU out of the parameters given as input variables
FAL_Pdu_Type	fal_pdu	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FALPDU types
LoadBuffer	Remote_dlsap_address, user_data	This function loads user data into the local buffer.
StartTransmitCycleTimer	arep_id	This function starts the timer specified by arep_id.

10 DLL mapping protocol machine (DMPM)

10.1 General

The DLL Mapping Protocol Machine is common to all the AREP types.

The primitives issued by ARPM to DMPM are passed to the data-link layer as the DLS primitives. The primitives issued to DMPM from the data-link layer are notified to an appropriate ARPM out of the ARPMs.

DMPM adds and deletes parameters to/from the primitives exchanged between ARPM and the data-link layer if necessary.

– Remarks about DL-identifiers:
The data-link layer specification defines two types of identifiers to distinguish each DL primitive or to match one DL outgoing primitive with the corresponding incoming primitive. These two identifiers are suffixed as DL-identifier and DLS-user-identifier, respectively. In a real implementation of an FAL-DL interface, these identifications may be achieved by means of a pointer to a memory location or a return value of a function call, or something else. For this reason, these identifiers are not included as parameters of the primitives issued by the ARPM.

The “DL-identifiers” and “DLS-user-identifiers” are mandatory in the DL-services. The FAL assumes that the values of these parameters are provided by a local means.

– Remark about DLS-user identification:
It is assumed that a connection between one ARPM instance and one DMPM instance is established locally rather than by means of a protocol. Therefore, DLS-user identification parameters are not used in the primitives issued by the ARPM.

– Remark about buffer or queue identifiers:
The data-link layer uses parameters to identify the queue or buffer shared between the data-link layer and the DLS-user. Although they are useful to clarify the operations of the data-link layer, none of them affects the protocol behaviour of the FAL and DL. In a real implementation, these parameters are implementation-dependent. Therefore, parameters that correspond direct to these buffer or queue identifiers are not described. A means for identifying the buffers and queues between the FAL and the DL is a local matter.

– Remark about initialization of the data-link layer:
The data-link layer specification defines services to setup resources within the layer, such as DL-Create or DL-Bind services. Although they are useful to clarify the operations of the data-link layer, none of them affects the protocol behavior of the FAL and DL. Therefore, the FAL assumes that such initialization procedures have been executed prior to the operations of the FAL state machines.

10.2 Primitive definitions

10.2.1 Primitives exchanged between DMPM and ARPM

Table 53 lists the primitives exchanged between the DMPM and the ARPM.

Table 53 – Primitives exchanged between DMPM and ARPM

Primitive name	Source	Associated parameters	Functions
Establish_req	ARPM	cardinality, remote_confirm, conveyance_policy, sequence_control	This primitive is used to request the establishment of a AR
Abort_req	ARPM		This primitive is used to request an abort without transferring an FAL-PDU.
FAL-PDU_req	ARPM	dlsap_id, called_address, dll_priority, dlsdu	This primitive is used to request the DMPM to transfer an FAL-PDU. It passes the FAL-PDU to the DMPM as a DLSDU. It also carries some of the data-link layer parameters that are referenced there.
Establish_cnf	DMPM	dlsap_id	This primitive is used to report completion of the requested establishment of an AR.
FAL-PDU_ind	DMPM	calling_address, fal_pdu,	This primitive is used to pass an FAL-PDU received as a data-link layer service data unit to a designated ARPM. It also carries some of the data-link layer parameters that are referenced in the ARPM.
FAL-PDU_cnf	DMPM	status	
Abort_ind	DMPM	reason	This primitive is used to convey the indication of abort of provider and its reason.
ErrorToARPM	DMPM	originator, reason	This primitive is used to convey selected communication errors reported by the data-link layer to a designated ARPM.

10.2.2 Primitives exchanged between data-link layer and ARPM

Table 54 lists the primitives exchanged between the data-link layer and the ARPM.

Table 54 – Primitives exchanged between data-link layer and DMPM

Primitive name	Source	Associated parameters	Functions
DL-UNITDATA _req	DMPM	dl_called address, dl_dls_user_data	
DL_CREATE_req	DMPM	Maximum DLSDU size, Maximum queue depth, Queue DL-identifier	
DL_BIND_req	DMPM	dl_service_subtype, dl_dlsap_id	
DL-DELETE_req	DMPM	Queue DL-identifier	
DL-UNBIND_req	DMPM	DLSAP DL-identifier	
DLM-SET_req	DMPM	DLM-object-identifier, Desired-value, dl_status	
DLM-GET_req	DMPM	DLM-object-identifier, Current-value, Status	
DLM-ACTION_req	DMPM	Desired-action	
DL-UNITDATA_ind	Data-link layer	dl_calling_address, dl_dls_user_data	
DL-UNITDATA_cnf	Data-link layer	dl_status	
DLM-ACTION_cnf	Data-link layer	dl_status	
DLM-EVENT_ind	Data-link layer	DLM-event-identifier	

10.2.3 Parameters of DMPM/data-link layer primitives

The parameters used with the primitives exchanged between the DMPM and the data-link layer are identical to those defined in Section 4 of this PAS. They are prefixed by "dl_" to indicate that they are used by the FAL.

10.3 DMPM state machine

10.3.1 DMPM states

The DMPM State Machine has only one possible state. The defined state and their descriptions are shown in Table 55 and Figure 18.

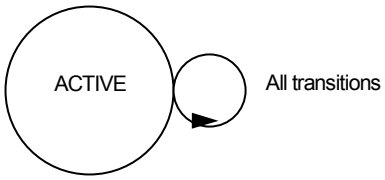


Figure 18 – State transition diagram of DMPM

Table 55 – DMPM states

State	Description
ACTIVE	The DMPM in the ACTIVE state is ready to transmit or receive primitives to or from the data-link layer and the ARPM.

10.3.2 DMPM state table

The DMPM state machine is described in Table 56 and Table 57

Table 56 – DMPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	Establish_req
		&& cardinality = "one-to-one"
		&& remote_confirm = "True"
		&& sequence_control := "True"
		=>
		DL_BIND_req(in){
		dl_service_subtype := "ASS"
		}
		
		DL_BIND_req(out) -- immediate response
		=>
		Establish_cnf{
		dlsap_id := dl_dlsap_id
		}
			ACTIVE

#	Current state	Event or condition => action	Next state

S2	ACTIVE	Establish_req
		&& cardinality = “one-to-one”
		&& remote_confirm = “True”
		&& sequence_control := “False”
		=>
		DL_BIND_req(in){
		dl_service_subtype := “AUS”
		}
		
		DL_BIND_req(out) -- immediate response
		=>
		Establish_cnf{
		dlsap_id := dl_dlsap_id
		}
			ACTIVE

S3	ACTIVE	Establish_req
		&& cardinality = “one-to-one”
		&& remote_confirm = “False”
		&& sequence_control := “False”
		=>
		DL_BIND_req(in){
		dl_service_subtype := “UUS”
		}
		
		DL_BIND_req(out) -- immediate response
		=>
		Establish_cnf{
		dlsap_id := dl_dlsap_id
		}
			ACTIVE

S4	ACTIVE	Establish_req
		&& cardinality = “one-to-many”
		&& remote_confirm = “False”
		&& sequence_control := “False”
		=>
		DL_BIND_req(in){
		dl_service_subtype := “MUS”
		}
		
		DL_BIND_req(out) -- immediate response
		=>
		Establish_cnf{
		dlsap_id := dl_dlsap_id
		}
			ACTIVE

S5	ACTIVE	Establish_req
		&& cardinality = “one-to-many”
		&& remote_confirm = “False”
		&& sequence_control := “True”
		=>
		DL_BIND_req(in){
		dl_service_subtype := “MSS”
		}
		
		DL_BIND_req(out) -- immediate response
		=>
		Establish_cnf{
		dlsap_id := dl_dlsap_id
		}
			ACTIVE

S6	ACTIVE	Abort_req => DL-UNBIND_req{}, Abort_ind{}	ACTIVE

S7	ACTIVE	FAL-PDU_req
		=>
		PickDlsap (dlsap_id),
		
		DL-UNITDATA _req{
		dl_called_address := called_address,
		dl_dls_user_data := dlsdu
		}
			ACTIVE

Table 57 – DMPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state
R1	ACTIVE	DL_Unitdata.ind && FindAREP (dl_called_address) = “False” => (no actions taken)	ACTIVE

R2	ACTIVE	DL_Unitdata.ind
		&& FindAREP (dl_called_address) = “True”
		=>
		FAL-PDU_ind {
		calling_address := dl_calling_address,
		fal_pdu := dl_dls_user_data
		}
			ACTIVE

R3	ACTIVE	DL_Unitdata.cnf
		&& dl_status <> “success”
		=>
		ErrorToARPM {
		originator := “local_dls”,
		reason := dl_status
		}
		FAL-PDU_cnf {
		status := dl_status
		}
			ACTIVE

R4	ACTIVE	DL_Unitdata.cnf
		&& dl_status = “success”
		=>
		(no actions taken)
		FAL-PDU_cnf {
		status := dl_status
		}
			ACTIVE

10.3.3 Functions used by DMPM

Table 58 contains the functions used by the DMPM, their arguments and their descriptions.

Table 58 – Functions used by the DMPM

Function name	Parameter	Description
PickDlsap	dlsap_id	This function selects the DLSAP specified by the dlsap_id parameter. After this function is executed, the attributes of the selected DLSAP are available to the state machine.
FindAREP	dl_called_address	This function identifies the AREP that shall be bound with an active DMPM. True means the AREP exists. After this function is executed, the attributes of the selected AREP are available to the state machine.

Bibliography

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

IEC 61158-3-17, *Industrial communication networks – Fieldbus specifications – Part 3-17: Data-link layer service definition – Type 17 elements*

IEC 61158-4-17, *Industrial communication networks – Fieldbus specifications – Part 4-17: Data-link layer protocol specification – Type 17 elements*

IEC 61784-1 (Ed.2.0), *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*

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-17: Spécification de protocole de la couche d'application – Éléments de Type 17

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 de la CEI 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 n'a prévu aucune procédure de marquage valant indication d'approbation et ne peut pas engager sa responsabilité pour les équipements déclarés conformes à une de ses Publications.
6) Il convient que tous les utilisateurs s'assurent 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.

NOTE L'utilisation de certains des types de protocole associés est restreinte par les détenteurs des droits de propriété intellectuelle correspondants. Dans tous les cas, 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 Liaison de données particulier avec des protocoles de couche physique et de couche d'application dans les combinaisons de types explicitement spécifiées dans la série CEI 61784. L'utilisation des divers types de protocole dans d'autres combinaisons peut nécessiter l'autorisation de leurs détenteurs de droits de propriété intellectuelle respectifs.

La CEI attire l'attention sur le fait qu'il est déclaré que la conformité à la présente norme peut impliquer l'utilisation des droits de propriété ci-dessous, la notation [xx] désignant le détenteur du droit associé:

Type 17:

Demande PCT n° PCT/JP2004/011537	[YEC]	Méthode de contrôle de la communication
Demande PCT n° PCT/JP2004/011538	[YEC]	Méthode de contrôle de la communication

La CEI ne prend pas position eu égard à la preuve, la validité et la portée de ces droits de propriété.

Les détenteurs de ces droits de propriété ont donné l'assurance à la CEI qu'ils consentent à négocier des licences avec des demandeurs du monde entier, en des termes et à des conditions raisonnables et non discriminatoires. A ce propos, la déclaration des détenteurs de ces droits de propriété est enregistrée à la CEI. Des informations peuvent être obtenues auprès de:

[YEC]: Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi, 180-8750 Tokyo,
180-8750 Tokyo,
Japon
Attn: Intellectual Property & Standardization Center

L'attention est attirée sur le fait que certains des éléments de la présente norme peuvent faire l'objet de droits de propriété autres que ceux mentionnés ci-dessus. La CEI ne doit pas être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

La Norme internationale CEI 61158-6-17 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.

Cette première édition et les parties de la sous-série CEI 61158-6 associées annulent et remplacent la CEI 61158-6:2003. L'édition de la présente partie constitue un ajout technique. La présente partie et les parties associées au Type 17 annulent et remplacent également la CEI/PAS 62405 publiée en 2005.

Cette édition de la CEI 61158-6 comporte les modifications importantes suivantes par rapport à l'édition précédente:

a) suppression de l'ancien bus de terrain de Type 6 pour défaut de pertinence de commercialisation;
b) ajout de nouveaux types de bus de terrain;
c) fractionnement de la Partie 6 de la troisième édition en plusieurs parties numérotées -6-2, -6-3, etc.

La présente version bilingue (2013-09) correspond à la version anglaise monolingue publiée en 2007-12.

Le texte anglais de cette norme est issu des documents 65C/476/FDIS et 65C/487/RVD.

Le rapport de vote 65C/487/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

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

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.

NOTE La révision de la présente norme sera synchronisée avec les autres parties de la série CEI 61158.

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

INTRODUCTION

La présente partie de la CEI 61158 s'inscrit dans une série créée pour faciliter l'interconnexion des composants de systèmes d'automation. Elle est liée à d'autres normes de la série définie par le modèle de référence des bus de terrain "à trois couches" décrit dans la CEI/TR 61158-1.

Le protocole d'application fournit le service d'application au moyen des services disponibles au niveau de la couche de 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, que doivent suivre les entités d'application (Application Entity, AE) homologues au moment de la communication. Ces règles de communication ont pour vocation de fournir une base de développement stable permettant d'atteindre différents objectifs:

- guider les développeurs et les concepteurs;
- réaliser les essais et acquérir l'équipement;
- établir un accord d'intégration des systèmes dans l'environnement de systèmes ouverts;
- améliorer la compréhension des communications en temps critique au sein de l'OSI.

La présente norme porte en particulier sur la communication et l'interfonctionnement des capteurs, des effecteurs et des autres appareils d'automation. Grâce à cette norme et à d'autres normes des modèles de référence OSI ou de bus de terrain, des systèmes par ailleurs incompatibles peuvent fonctionner ensemble, quelle que soit leur combinaison.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-17: Spécification de protocole de la couche d'application – Éléments de Type 17

1 Domaine d'application

1.1 Généralités

La couche d'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. A cet égard, la couche FAL peut être considérée comme une "fenêtre entre programmes d'application correspondants".

La présente norme fournit des éléments communs pour les communications de messagerie en temps critique ou non entre des programmes d'application dans un environnement et avec un matériel d'automation spécifiques aux bus de terrain de Type 17. L'expression "en temps critique" signale l'existence d'une fenêtre temporelle dans laquelle des actions spécifiées doivent être exécutées, avec un niveau de certitude défini. 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 spécifie les interactions entre les applications distantes et définit le comportement, visible par un observateur externe, assuré par la couche d'application de bus de terrain de Type 17, en termes

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

b) de syntaxe de transfert définissant les règles de codage qui s'appliquent aux unités de données de protocole de la couche d'application;

c) de diagramme d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication;

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

La présente norme vise à définir le protocole mis en place pour:

1) définir la représentation filaire des primitives de service définies dans la CEI 61158-5-17, et

2) définir le comportement visible par un observateur externe associé à leur transfert.

La présente norme spécifie le protocole de la couche d'application de bus de terrain de Type 17, en conformité avec le modèle de référence de base OSI (ISO/CEI 7498) et avec la structure de la couche Application OSI (ISO/CEI 9545).

1.2 Spécifications

La présente norme a pour objectif principal de spécifier la syntaxe et le comportement du protocole de la couche d'application qui transmet les services de la couche d'application définis dans la CEI 61158-5-17.

Un objectif secondaire consiste à fournir des chemins de migration à partir des protocoles de communication industriels antérieurs. Ce dernier objectif explique la diversité des protocoles normalisés dans la série CEI 61158-6.

1.3 Conformité

La présente norme ne spécifie pas de mises en œuvre ni de produits particuliers, pas plus qu'elle ne limite les mises en œuvre des entités de la couche d'application dans les systèmes d'automation industriels. La conformité est obtenue par le biais de la mise en œuvre de cette spécification de protocole de la couche d'application.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. 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).

IEC 61158-5-17, *Industrial communication networks – Fieldbus specifications – Part 5-17: Application layer service definition – Type 17 elements* (disponible en anglais uniquement)

ISO/CEI 7498 (toutes les parties), *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base*

ISO/CEI 8824-2, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): spécification des objets informationnels*

ISO/CEI 8825-1, *Technologies de l'information – Règles de codage ASN.1: Spécification des règles de codage de base (BER), des règles de codage canoniques (CER) et des règles de codage distinctives (DER)*

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

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

3 Définitions

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

3.1 Termes et définitions

3.1.1 Termes 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) entité d'application

b) unité de données de protocole d'application

c) élément de service d'application

3.1.2 Termes de l'ISO/CEI 8824-2

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

a) type any (indifférent)

b) type bitstring (chaîne binaire)

c) type Boolean (booléen)

d) type choice (choix)

e) false (faux)

f) type integer (entier)
g) type null (vide)
h) type octetstring (chaîne d'octets)
i) type sequence of (séquence de)
j) type sequence (séquence)
k) type simple
l) type structuré
m) type tagged (étiqueté)
n) true (vrai)
o) type
p) valeur

3.1.3 Termes de l'ISO/CEI 10731

a) (N)-connection (connexion (N))
b) (N)-entity (entité (N))
c) (N)-layer (couche (N))
d) (N)-service (service (N))
e) (N)-service-access-point (point d'accès au service (N))
f) (primitive) confirm (confirmation)
g) (primitive) indication
h) (primitive) request (demande)
i) (primitive) response (réponse)

3.1.4 Autres termes et définitions

3.1.4.1
application
fonction ou structure de données pour laquelle des données sont consommées ou produites

3.1.4.2
processus d'application
partie d'une application distribuée sur un réseau, qui est située sur un appareil et adressée sans ambiguïté

3.1.4.3
relation d'applications
association de type coopératif entre deux invocations d'entités d'application (application-entity-invocation) ou plus, dans un but d'échange d'informations et de coordination de leur action conjointe

NOTE Cette relation est activée soit par l'échange d'unités de données de protocole d'application (application-protocol-data-unit), soit à la suite d'activités de préconfiguration.

3.1.5
élément de service d'application de relation d'applications
élément de service d'application (application-service-element) qui fournit le seul moyen d'établir et de rompre toute relation d'applications

3.1.5.1
point d'extrémité de relation d'applications
contexte et comportement d'une relation d'applications, vus et maintenus par un des processus d'application impliqués dans la relation d'applications

NOTE Chaque processus d'application impliqué dans la relation d'applications maintient son propre point d'extrémité de relation d'applications.

3.1.5.2
attribut
description d'une caractéristique ou fonction, visible par un observateur externe, d'un objet

NOTE Les attributs d'un objet contiennent des informations sur les portions variables d'un objet. En règle générale, ils fournissent des informations de statut ou régissent l'action d'un objet. Les attributs peuvent également influer sur le comportement d'un objet. Les attributs se répartissent en deux catégories: les attributs de classe et les attributs d'instance.

3.1.5.3
comportement
indication de la manière dont un objet réagit à des événements particuliers

3.1.5.4
pont
équipement intermédiaire qui permet de connecter au moins deux segments grâce à une fonction de relais de couche de liaison de données

3.1.5.5
canal
liaison physique ou logique unique d'un objet d'application d'entrée ou de sortie d'un serveur avec le processus

3.1.5.6
classe
ensemble d'objets représentant le même type de composant du système

NOTE Une classe est une généralisation d'un objet, un modèle qui permet de définir des variables et des méthodes. Tous les objets d'une classe possèdent une forme et un comportement identiques, mais leurs attributs contiennent généralement des données différentes.

3.1.5.7
client
a) objet qui utilise les services d'un autre objet (serveur) pour réaliser une tâche

b) initiateur d'un message auquel un serveur réagit

3.1.5.8
connexion
liaison logique entre deux objets d'application qui peuvent se trouver au sein du même appareil ou dans des appareils différents

NOTE 1 Les connexions peuvent être soit point à point, soit multipoint.

NOTE 2 La liaison logique entre puits et source d'attributs et de services au niveau des différentes interfaces personnalisées des éléments ASE RT-Auto est appelée "interconnexion". Une distinction est faite entre interconnexion de données et interconnexion d'événements. L'ensemble constitué de la liaison logique et du flux de données entre le puits et la source d'éléments de données d'automation est appelé "interconnexion de données". L'ensemble constitué de la liaison logique et du flux de données entre le puits (méthode) et la source (événement) de services opérationnels est appelé "interconnexion d'événements".

3.1.5.9
point de connexion
tampon qui est représenté comme étant une sous-instance d'un objet Assembly (ensemble)

3.1.5.10
chemin de transport
flux unidirectionnel d'unités APDU à travers une relation d'applications

3.1.5.11
AR dédiée
AR directement utilisée par l'utilisateur FAL

NOTE Dans les AR dédiées, seuls l'en-tête FAL et les données d'utilisateur sont transférés.

3.1.5.12
appareil
matériel physique connecté à la liaison

NOTE Un appareil peut contenir plusieurs nœuds.

3.1.5.13
domaine
portion du réseau RTE composée d'un ou deux sous-réseaux

NOTE Deux sous-réseaux sont nécessaires pour former un réseau RTE à double redondance et chaque nœud d'extrémité du domaine est connecté à ces deux sous-réseaux.

3.1.5.14
maître de domaine
poste chargé du diagnostic des voies menant à tous les autres domaines, de la distribution du temps réseau aux nœuds situés à l'intérieur du domaine, de l'acquisition du temps absolu auprès du maître de temps réseau et de la notification du statut du domaine

3.1.5.15
numéro de domaine
identificateur numérique qui désigne un domaine

3.1.5.16
nœud d'extrémité
nœud producteur ou consommateur

3.1.5.17
point d'extrémité
une des entités en communication impliquées dans une connexion

3.1.5.18
erreur
divergence entre une valeur ou condition calculée, observée ou mesurée et la valeur ou condition spécifiée ou théoriquement correcte

3.1.5.19
classe d'erreurs
groupement général de définitions d'erreurs proches et des codes d'erreur associés

3.1.5.20
pont externe
pont qui n'est directement connecté à aucun pont interne ni à aucun poste RTE

3.1.5.21
événement
instance d'un changement de conditions

3.1.5.22
groupe
a) <généralités> terme général désignant un ensemble d'objets. Usages spécifiques:

b) <adressage> dans la description d'une adresse, adresse qui identifie plusieurs entités

3.1.5.23
interface
a) frontière commune entre deux unités fonctionnelles, définie par des caractéristiques fonctionnelles, des caractéristiques de signal ou d'autres caractéristiques adaptées

b) ensemble d'attributs et de services de classe FAL représentant une vue spécifique de la classe FAL

3.1.5.24
port d'interface
point de connexion physique d'un nœud d'extrémité, qui possède une adresse DL indépendante

3.1.5.25
pont interne
pont qui n'est directement connecté à aucun routeur, pont externe ou nœud non conforme à la présente spécification

3.1.5.26
invocation
acte d'utiliser un service ou une autre ressource d'un processus d'application

NOTE Chaque invocation représente un fil de commande distinct qui peut être décrit par son contexte. Une fois le service terminé ou la ressource libérée, l'invocation cesse d'exister. Dans le cas des invocations de services, un service lancé mais pas encore terminé est appelé "invocation de services en cours". Dans ce même cas, un ID d'invocation peut être utilisé pour identifier sans ambiguïté l'invocation de services et la différencier des autres invocations de services en cours.

3.1.5.27
pont de jonction
pont qui est connecté à au moins un routeur, pont externe ou nœud non conforme à la présente spécification et à au moins un pont interne ou poste RTE

3.1.5.28
liaison
canal de communication physique qui sépare deux nœuds

3.1.5.29
méthode
<objet> synonyme d'un service opérationnel délivré par l'élément ASE serveur et appelé par un client

3.1.5.30
réseau
ensemble de nœuds reliés par un support de communication d'un type ou d'un autre, avec d'éventuels répéteurs, ponts, routeurs et passerelles de couche inférieure intermédiaires

3.1.5.31
maître de temps réseau
poste qui distribue le temps réseau aux maîtres de domaine

3.1.5.32
nœud
entité DL unique telle qu'elle se présente sur une liaison locale

3.1.5.33
nœud d'interface non redondant
nœud qui possède un seul port d'interface

3.1.5.34
poste non redondant
poste composé d'un seul nœud d'extrémité

NOTE Les expressions “poste non redondant” et “nœud d'extrémité” sont synonymes.

3.1.5.35
objet
représentation abstraite d'un composant particulier dans un appareil; il s'agit généralement d'un ensemble de données (sous la forme de variables) et de méthodes (procédures) associées, destiné à l'exploitation des données dont l'interface et le comportement sont clairement définis

3.1.5.36
émetteur
client chargé de l'établissement d'un chemin de connexion vers la cible

3.1.5.37
chemin
canal de communication logique qui sépare deux nœuds et se compose d'une ou deux liaisons

3.1.5.38
homologue
rôle d'un point d'extrémité d'AR dans lequel il est capable d'agir à la fois comme client et comme serveur

3.1.5.39
producteur
nœud chargé de l'envoi des données

3.1.5.40
fournisseur
source d'une connexion de données

3.1.5.41
éditeur
rôle d'un point d'extrémité d'AR qui transmet des unités APDU au bus de terrain afin qu'elles soient consommées par un ou plusieurs abonnés

NOTE Un éditeur peut ne pas connaître l'identité des abonnés ni leur nombre; il peut publier ses unités APDU au moyen d'une AR dédiée.

3.1.5.42
nœud d'interface redondant
nœud doté de deux ports d'interface dont l'un est connecté à un réseau primaire et l'autre à un réseau secondaire

3.1.5.43
poste redondant
poste composé d'une paire de nœuds d'extrémité

NOTE Les nœuds d'extrémité d'un poste redondant possèdent tous le même numéro de poste, mais des adresses DL différentes.

3.1.5.44
ressource
capacité de traitement ou d'information d'un sous-système

3.1.5.45
poste RTE
poste conforme à la présente spécification

3.1.5.46
voie
canal de communication logique qui sépare deux nœuds d'extrémité de communication

3.1.5.47
routeur
équipement intermédiaire qui permet de connecter au moins deux sous-réseaux grâce à une fonction de relais de couche Réseau

3.1.5.48
segment
canal de communication qui permet de connecter directement deux nœuds, sans faire intervenir de pont

3.1.5.49
serveur
a) rôle d'un AREP dans lequel il renvoie une unité APDU de réponse de service confirmé au client à l'origine de la demande

b) objet qui offre des services à un autre objet (client)

3.1.5.50
service
action ou fonction qu'un objet et/ou une classe d'objets exécutent à la demande d'un autre objet et/ou une autre classe d'objets

3.1.5.51
poste
nœud d'extrémité ou paire de nœuds d'extrémité qui accomplit une fonction d'application spécifique

3.1.5.52
numéro de poste
identificateur numérique qui désigne un poste RTE

3.1.5.53
sous-réseau
portion d'un réseau qui ne comporte aucun routeur. Un sous-réseau se compose de nœuds d'extrémité, de ponts et de segments

NOTE Les nœuds d'extrémité inclus dans un sous-réseau possèdent tous la même adresse réseau IP.

3.1.5.54
abonné
rôle d'un AREP dans lequel il reçoit les unités APDU produites par un éditeur

3.2 Abréviations et symboles

3.2.1 Abréviations de l'ISO/CEI 10731

ASE	élément de service d'application (Application-Service-Element)
OSI	interconnexion de systèmes ouverts (Open Systems Interconnection)

3.2.2 Abréviations et symboles de l'ISO/CEI 7498-1

DL- (comme préfixe) couche de liaison de données (Data-Link layer)

DLL couche DL (DL-Layer)

DLM gestion DL (DL-Management)

DLS service DL (DL-Service)

DLSAP point d'accès au service DL (DL-Service-Access-Point)

DLSDU unité de données de service DL (DL-Service-Data-Unit)

3.2.3 Abréviations et symboles de la CEI 61158-5-17

AE entité d'application (Application Entity)

AL couche d'application (Application Layer)

AP processus d'application (Application Process)

APDU unité de données de protocole d'application (Application Protocol Data Unit)

AR relation d'applications (Application Relationship)

AREP point d'extrémité de relation d'applications (Application Relationship EndPoint)

ASN.1 notation de syntaxe abstraite numéro un (Abstract Syntax Notation one)

BCD décimal codé binaire (Binary Coded Decimal)

Cnf confirmation

cnf primitive de confirmation

Ev_ préfixe des types de données définis pour l'élément ASE d'événement

FAL couche Application de bus de terrain (Fieldbus Application Layer)

Gn_ préfixe des types de données définis pour un usage général

ID identificateur

CEI Commission Electrotechnique Internationale

Ind indication

ind primitive d'indication

IP protocole Internet (Internet Protocol)

ISO Organisation internationale de normalisation

lsb bit de poids faible (least significant bit)

msb bit de poids fort (most significant bit)

PDU unité de données de protocole (Protocol Data Unit)

Req demande (Request)

req primitive de demande (request)

Rsp réponse (Response)

rsp primitive de réponse (response)

SAP point d'accès au service (Service Access Point)

SDU unité de données de service (Service Data Unit)

3.2.4 Autres abréviations et symboles

ARPM machine de protocole de relations d'applications (Application Relationship Protocol Machine)

FSPM machine de protocole de service FAL (FAL Service Protocol Machine)

MSU-AR AREP éditeur/abonné de réseau multipoint planifié non confirmé (Multipoint network-Scheduled Unconfirmed publisher/subscriber

AREP)

MTU-AR AREP éditeur/abonné d'utilisateur multipoint déclenché non confirmé (Multipoint user-Triggered Unconfirmed publisher/subscriber AREP)

PSU-AR AREP client/serveur de réseau point à point planifié non confirmé (Point-to-point network-Scheduled Unconfirmed client/server AREP)

PTC-AR AREP client/serveur d'utilisateur point à point déclenché confirmé (Point-to-point user-Triggered Confirmed client/server AREP)

PTU-AR AREP client/serveur d'utilisateur point à point déclenché non confirmé (Point-to-point user-Triggered Unconfirmed client/server AREP)

3.3 Conventions

3.3.1 Conventions générales

La présente norme utilise les conventions de description énoncées dans l'ISO/CEI 10731.

La présente norme utilise les conventions de description énoncées dans la sous-série CEI 61158-5 pour les définitions de service FAL.

3.3.2 Conventions relatives aux définitions de syntaxe abstraite des unités APDU

La présente norme utilise les conventions de description énoncées dans l'ISO/CEI 8824-2 pour les définitions d'unité APDU.

3.3.3 Conventions relatives aux définitions de syntaxe de transfert des unités APDU

La présente norme utilise les conventions de description énoncées dans l'ISO/CEI 8825-1 pour les définitions de syntaxe de transfert.

3.3.4 Conventions relatives aux définitions de diagramme d'états d'entité AE

Le Tableau 1 décrit les conventions utilisées pour les définitions de diagramme d'états d'entité AE.

Tableau 1 – Conventions utilisées pour les définitions de diagramme d'états d'entité AE

N°	État actuel	Événement/condition => action	État suivant
Nom de ce passage	État actuel auquel s'applique ce passage d'état	Événements ou conditions déclenchant ce passage d'état. => Actions effectuées lorsque les événements ou conditions ci-dessus sont réunis. Elles figurent toujours au-dessous des événements ou conditions et sont toujours présentées avec un retrait	État suivant dans lequel passe le diagramme d'états une fois les actions de ce passage effectuées

Les conventions utilisées dans les descriptions d'événements, de conditions et d'actions sont les suivantes:

:= La valeur d'un élément, à gauche, est remplacée par la valeur d'un élément, à droite. Si un élément à droite est un paramètre, il provient de la primitive indiquée comme événement d'entrée.

xxx Indique le nom du paramètre.

Exemple:

Identifier := reason

signifie que la valeur du paramètre "reason" est attribuée au paramètre appelé "Identifier".

"xxx" Indique une valeur fixe.

Exemple:

Identifier := “abc”

signifie que la valeur "abc" est attribuée à un paramètre appelé "Identifier".

= Condition logique indiquant qu'un élément à gauche est égal à un élément à droite.

< Condition logique indiquant qu'un élément à gauche est inférieur à l'élément à droite.

> Condition logique indiquant qu'un élément à gauche est supérieur à l'élément à droite.

<> Condition logique indiquant qu'un élément à gauche est différent d'un élément à droite.

&& Indique le "ET" logique.

|| Indique le "OU" logique.

La séquence d'actions et les actions de remplacement peuvent être exécutées au moyen des mots réservés suivants:

for

endfor

if

else

elseif

Les exemples ci-dessous décrivent l'utilisation des mots réservés.

Exemple 1:

```
for (Identifier := valeur_début to valeur_fin)
        actions
endfor
```

Exemple 2:

```
If (condition)
        actions
else
        actions
endif
```

4 Description de la syntaxe abstraite

4.1 Syntaxe abstraite des unités PDU de couche FAL

4.1.1 Définition de premier niveau

```
FalArPDU ::=
          ConfirmedSend-CommandPDU
        || ConfirmedSend-ResponsePDU
        || UnconfirmedSend-CommandPDU
```

4.1.2 En-tête FalArHeader

```
FalArHeader ::= Unsigned8{
        -- bits 8 à 7      ProtocolVersion
        -- bits 6 à 4      ProtocolIdentifier
        -- bits 3 à 1      PDUIdentifier
}
```

4.1.3 Service d'envoi confirmé

```
ConfirmedSend-CommandPDU ::= SEQUENCE {
        FalArHeader,
        ServiceType
        InvokeID,
        ConfirmedServiceRequest
}

ConfirmedSend-ResponsePDU ::= SEQUENCE {
        FalArHeader,
        ServiceType
        InvokeID,
        ConfirmedServiceResponse
}
```

4.1.4 Service d'envoi non confirmé

```
UnconfirmedSend-CommandPDU ::= SEQUENCE {
        FalArHeader,
        ServiceType
        InvokeID,
        UnconfirmedServiceRequest
}
```

4.2 Syntaxe abstraite du corps des unités PDU

4.2.1 Unités PDU de demande de service confirmé

```
ConfirmedServiceRequest ::= CHOICE {
        Read-Request            [0]   IMPLICIT   Read-RequestPDU,
        Write-Request           [1]   IMPLICIT   Write-RequestPDU,
        DownLoad-Request        [2]   IMPLICIT   DownLoad-RequestPDU,
        UpLoad-Request          [3]   IMPLICIT   UpLoad-RequestPDU,
        Start-Request           [4]   IMPLICIT   Start-RequestPDU,
        Stop-Request            [5]   IMPLICIT   Stop-RequestPDU,
        Resume- Request         [6]   IMPLICIT   Resume-RequestPDU,
        DelayCheck-Request      [7]   IMPLICIT   Time- RequestPDU,
}
```

4.2.2 Unités PDU de réponse de service confirmé

```
ConfirmedServiceResponse ::= CHOICE {
        Read-Response                   [0]     IMPLICIT    Read-ResponsePDU,
        Write-Response                  [1]     IMPLICIT    Write-ResponsePDU,
        DownLoad-Response               [2]     IMPLICIT    DownLoad-ResponsePDU,
        UpLoad-Response                 [3]     IMPLICIT    UpLoad-ResponsePDU,
        Start-Response                  [4]     IMPLICIT    Start-ResponsePDU,
        Stop-Response                   [5]     IMPLICIT    Stop-ResponsePDU,
        Resume-Response                 [6]     IMPLICIT    Resume-ResponsePDU,
        DelayCheck-Response             [7]     IMPLICIT    Time-ResponsePDU
}
```

4.2.3 Unités PDU non confirmées

```
UnconfirmedServiceRequest ::= CHOICE {
        InformationReport-Request       [0]     IMPLICIT    InformationReport-RequestPDU,
        EventNotification-Request       [1]     IMPLICIT    EventNotification-RequestPDU,
        EventRecovery-Request           [2]     IMPLICIT    EventRecovery-RequestPDU,
        TimeDistribution-Request        [3]     IMPLICIT    TimeDistribute-RequestPDU,
        SetTime-Request                 [4]     IMPLICIT    SetTime-RequestPDU,
        InDiag-Request                  [5]     IMPLICIT    InDiag-RequestPDU,
        ExDiag-Request                  [6]     IMPLICIT    ExDiag-RequestPDU,
        StationStatusReport-Request     [7]     IMPLICIT    StationStatusReport-RequestPDU,
        DomainStatusReport-Request      [8]     IMPLICIT    DomainStatusReport-RequestPDU
}
```

4.2.4 Informations d'erreur

4.2.4.1 Type d'erreur

```
ErrorType ::= SEQUENCE {
        errorClass                      [0]     IMPLICIT    ErrorClass,
        additionalCode                  [1]     IMPLICIT    Integer16 OPTIONAL,
        additionalDescription           [2]     IMPLICIT    VisibleString OPTIONAL,
        additionalInfo                  [3]     IMPLICIT    ANY OPTIONAL
}
```

4.2.4.2 Classe d'erreurs

```
ErrorClass ::= CHOICE {
        noError                         [0]     IMPLICIT    Integer8 {
                                                    normal                          (0),
                                                    other                           (1)
        }
        applicationReference            [1]     IMPLICIT    Integer8 {
                                                    other                           (0),
                                                    application-unreachable         (1),
                                                    application-reference-invalid   (2),
                                                    context-unsupported             (3)
        }
        definition                      [2]     IMPLICIT    Integer8 {
                                                    other                           (0),
                                                    object-undefined                (1),
                                                    object-attributes-inconsistent  (2),
                                                    name-already-exists             (3),
                                                    type-unsupported                (4),
                                                    type-inconsistent               (5)
        }
        resource                        [3]     IMPLICIT    Integer8 {
                                                    other                           (0),
                                                    memory-unavailable              (1)
        }
        service                         [4]     IMPLICIT    Integer8 {
                                                    other                           (0),
                                                    object-state-conflict           (1),
                                                    pdu-size                        (2),
                                                    object-constraint-conflict      (3),
                                                    parameter-inconsistent          (4),
                                                    illegal-parameter               (5)
        }
        access                          [5]     IMPLICIT    Integer8 {
                                                    other                           (0),
                                                    object-invalidated              (1),
                                                    hardware-fault                  (2),
                                                    object-access-denied            (3),
                                                    invalid-address                 (4),
                                                    object-attribute-inconsistent   (5),
                                                    object-access-unsupported       (6),
                                                    object-non-existent             (7),
                                                    type-conflict                   (8),
                                                    named-access-unsupported        (9),
                                                    access-to-element-unsupported   (10)
        }
        conclude                        [6]     IMPLICIT    Integer8 {
                                                    other                           (0)
        }
        other                           [7]     IMPLICIT    Integer8 {
                                                    other                           (0)
        }
}
```

4.3 Unités PDU destinées aux éléments ASE

4.3.1 Unités PDU destinées à l'élément ASE de variable

4.3.1.1 Unités PDU de service de lecture

```
Read-RequestPDU ::= SEQUENCE {
        objectSpecifier CHOICE{
            variableSpecifier                   Gn_KeyAttribute,
            variableListSpecifier               Gn_KeyAttribute,
            listOfvariable                      SEQUENCE OF Gn_KeyAttribute
        }
        optionalParameters          [0]     IMPLICIT    ANY OPTIONAL
}
```

```
Read-ResponsePDU ::= SEQUENCE {
        result CHOICE{
              accessStatus                [0]    IMPLICIT    ErrorType,
              listOfAccessStatus          [1]    IMPLICIT    SEQUENCE OF ErrorType
        }
        value CHOICE{
              data                        [0]    IMPLICIT    ANY,
              listOfData                  [1]    IMPLICIT    SEQUENCE OF ANY
        }
        variableType CHOICE{
              dataType                    [0]    IMPLICIT    Gn_FullyNestedTypeDescription OPTIONAL,
              listOfDataType              [1]    IMPLICIT    SEQUENCE OF Gn_FullyNestedTypeDescription
                                                             OPTIONAL
        }
        optionalParameters                [0]    IMPLICIT    ANY OPTIONAL
}
```

4.3.1.2 Unités PDU de service d'écriture

```
Write-RequestPDU ::= SEQUENCE {
        objectSpecifier CHOICE{
              variableSpecifier                                Gn_KeyAttribute,
              variableListSpecifier                            Gn_KeyAttribute,
              listOfVariable                                   SEQUENCE OF Gn_KeyAttribute
        }
        variableType CHOICE{
              dataType                    [0]    IMPLICIT    Gn_FullyNestedTypeDescription OPTIONAL,
              listOfDataType              [1]    IMPLICIT    SEQUENCE OF Gn_FullyNestedTypeDescription
                                                             OPTIONAL
        }
        value CHOICE{
              data                        [0]    IMPLICIT    ANY,
              listOfData                  [1]    IMPLICIT    SEQUENCE OF ANY
        }
        optionalParameters                [0]    IMPLICIT    ANY OPTIONAL
}

Write-ResponsePDU ::= SEQUENCE {
        result CHOICE{
              accessStatus                [0]    IMPLICIT    ErrorType,
              listOfAccessStatus          [1]    IMPLICIT    SEQUENCE OF ErrorType
        }
        optionalParameters                [0]    IMPLICIT    ANY OPTIONAL
}
```

4.3.1.3 Unités PDU de service de transmission d'informations

```
InformationReport-RequestPDU::= SEQUENCE {
        ListOfVariableSpecifier CHOICE {
              variableListSpecifier                            Gn_KeyAttribute,
              listOfVariable                                   SEQUENCE OF Gn_KeyAttribute
        },
        listOfDataType                [1]    IMPLICIT    SEQUENCE OF Gn_FullyNestedTypeDescription
                                                         OPTIONAL
        listOfData                    [2]    IMPLICIT    SEQUENCE OF ANY
        optionalParameters            [3]    IMPLICIT    ANY OPTIONAL
}
```

4.3.2 Unités PDU destinées à l'élément ASE d'événement

4.3.2.1 Service de notification d'événement

```
EventNotification-RequestPDU ::= SEQUENCE {
      eventNotifierID                        IMPLICIT    Gn_ KeyAttribute,,
      notificvationSequenceNumber   [1]    IMPLICIT    Ev_SequenceNumber,
      listOfEvent                   [2]    IMPLICIT    SEQUENCE OF Ev_EventData,
      Notification Time             [3]    IMPLICIT    Ev_TimeTag OPTIONAL
      optionalParameters            [4]    IMPLICIT    ANY OPTIONAL
}
```

4.3.2.2 Service de récupération de notification

```
EventRecovery-RequestPDU ::= SEQUENCE {
        eventNotifierID                         IMPLICIT    Gn_ KeyAttribute,,
        sequenceNumber                  [1]     IMPLICIT    Ev_SequenceNumber OPTIONAL
}
```

4.3.3 Unités PDU destinées à l'élément ASE de région de charge

4.3.3.1 Service de téléchargement

```
DownLoad-RequestPDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        segmentIdentifier               [1]     IMPLICIT    ANY,
        loadData                        [2]     IMPLICIT    octetString,
}


DownLoad-ResponsePDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        result                          [1]     IMPLICIT    ErrorType,
}
```

4.3.3.2 Service de chargement

```
UpLoad-RequestPDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        segmentIdentifier               [1]     IMPLICIT    ANY,
}


UpLoad-ResponsePDU ::= SEQUENCE {
        loadRegionKeyAttribute                              Gn_KeyAttribute,
        result                          [1]     IMPLICIT    ErrorType,
        loadData                        [2]     IMPLICIT    octetString,
}
```

4.3.4 Unités PDU destinées à l'élément ASE d'invocation de fonctions

4.3.4.1 Service de démarrage

```
Start-RequestPDU ::= SEQUENCE {
        keyAttribute                                        Gn_KeyAttribute,
        optionalParameters              [1]     IMPLICIT    ANY OPTIONAL
}


Start-ResponsePDU ::= ErrorType
```

4.3.4.2 Service d'arrêt

```
Stop-RequestPDU ::= SEQUENCE {
        keyAttribute                                        Gn_KeyAttribute,
        optionalParameters              [1]     IMPLICIT    ANY OPTIONAL
}


Stop-ResponsePDU ::= ErrorType
```

4.3.4.3 Services de reprise

```
Resume-RequestPDU ::= SEQUENCE {
        keyAttribute                                        Gn_KeyAttribute,
        optionalParameters              [1]     IMPLICIT    ANY OPTIONAL
}


Resume-ResponsePDU ::= ErrorType
```

4.3.5 Unités PDU destinées à l'élément ASE de temps

4.3.5.1 Service de temps

```
Time-RequestPDU ::= Time-PDU

Time-ResponsePDU ::= Time-PDU

TimeDistribute-RequestPDU ::= Time-PDU

Time-PDU ::= SEQUENCE {
        timeControl             [0]   IMPLICIT   Tm_TimeControl,
        Stratum                 [1]   IMPLICIT   Unsigned8
        PollInterval            [2]   IMPLICIT   Tm_TimeValue1,
        Precision               [3]   IMPLICIT   Tm_TimeValue1,
        rootDelay               [4]   IMPLICIT   Tm_TimeValue2,
        rootDispersion          [5]   IMPLICIT   Tm_TimeValue2,
        referenceIdentifier     [6]   IMPLICIT   Tm_ReferenceID,
        referenceTimestamp      [7]   IMPLICIT   Tm_Time,
        originateTimestamp      [8]   IMPLICIT   Tm_Time,
        receiveTimestamp        [9]   IMPLICIT   Tm_Time,
        transmitTimestamp       [10]  IMPLICIT   Tm_Time,
}

SetTime-RequestPDU ::= SEQUENCE {
        timeValue               [0]   IMPLICIT   Tm_Time,
        optionalParameters      [1]   IMPLICIT   ANY OPTIONAL
}
```

4.3.6 Unités PDU destinées à l'élément ASE de gestion de réseau

4.3.6.1 Service de gestion de réseau

```
InDiag-RequestPDU ::= SEQUENCE {
        nodeInformation         [0]   IMPLICIT   Nm_NodeInformation,
        nodeStatus              [1]   IMPLICIT   Nm_NodeStatus,
        nodePublicKey           [2]   IMPLICIT   Nm_PublicKey,
        llistOfPathStatus       [3]   IMPLICIT   Nm_ListOfPathStatus
}

ExDiag-RequestPDU ::= SEQUENCE {
        doaminInformation       [0]   IMPLICIT   Nm_DoaminInformation,
        domainStatus            [1]   IMPLICIT   Nm_DoaminStatus,
        domainPublicKey         [2]   IMPLICIT   Nm_PublicKey,
        masterPriority          [3]   IMPLICIT   Unsigned8,
        llistOfPathStatus       [4]   IMPLICIT   Nm_ListOfPathStatus,
        listOfNodeStatus        [5]   IMPLICIT   SEQUENCE OF Nm_NodeStatus
}

StationStatusReport-RequestPDU ::= SEQUENCE {
        nodeInformation         [0]   IMPLICIT   Nm_NodeInformation,
        nodeStatus              [1]   IMPLICIT   Nm_NodeStatus
}

DomainStatusReport-RequestPDU ::= SEQUENCE {
        doaminInformation       [0]   IMPLICIT   Nm_DoaminInformation,
        domainStatus            [1]   IMPLICIT   Nm_DomainStatus
}
```

4.4 Définition des types

4.4.1 Types d'élément ASE de variable

Il n'existe aucun type spécial pour l'élément ASE de variable.

4.4.2 Types d'élément ASE d'événement

```
Ev_SequenceNumber ::= Unsigned8

Ev_EventData ::= ANY

En_EventCount ::= Unsigned8

Ev_TimeTag ::= Unsigned16
```

4.4.3 Types d'élément ASE de région de charge

Il n'existe aucun type spécial pour l'élément ASE de région de charge.

4.4.4 Types d'élément ASE d'invocation de fonctions

Il n'existe aucun type spécial pour l'élément ASE d'invocation de fonctions.

4.4.5 Types d'élément ASE de temps

```
Tm_TimeControl ::= BitString8 {
        -- bits 8 à 7      LeapIndidator
        -- bits 6 à 4      ProtocolVersion
        -- bits 3 à 1      TimeMode
}

Tm_TimeValue1 ::= Unsigned32        -- entier signé de 8 bits, en secondes par rapport à la puissance de 2 la
                                    plus proche

Tm_TimeValue2 ::= Unsigned32        -- nombre en virgule fixe signé de 32 bits, en secondes,
                                    -- la virgule fractionnaire étant placée entre le bit 15 et le bit 16

Tm_ReferenceID ::= VisibleString4   -- identifie la source de référence concernée

Tm_Time ::= SEQUENCE{
      Seconds              [0]          Unsigned32
      SecondsFraction      [2]          Unsigned32
}
```

4.4.6 Types d'élément ASE de gestion de réseau

```
Nm_NodeInformation ::= SEQUENCE {
      NodeIdentifier                  [0]   IMPLICIT   Nm_NodeIdentifier,
      NoOfInterfaces                  [1]   IMPLICIT   Integer8,
      InterfaceID                     [2]   IMPLICIT   Unsigned8,
      PerformanceClass SEQUENCE {
          MasterPriority              [11]  IMPLICIT   Unsigned8,
          TransmissionClass           [12]  IMPLICIT   Unsigned8,
          ResponseClass               [13]  IMPLICIT   Unsigned8,
          TimePrecisionLevel          [14]  IMPLICIT   Unsigned8,
      }
      configurationSUM                [4]   IMPLICIT   Unsigned32,
      localNodeTime                   [5]   IMPLICIT   Tm_Time,
      diagInterval                    [6]   IMPLICIT   BinaryTime2,
      stationCoefficeincy             [7]   IMPLICIT   Unsigned16
}

Nm_NodeStatus ::= BitString8 {
        -- bit 8        CPU-Status                  -- True: prêt, False: non prêt
        -- bit 7        communication-status        -- True: prêt, False: non prêt
        -- bit 6        reserved-status             -- True: réservé, False: non réservé
        -- bit 5        redundancy-status           -- True: en service, False: en attente
        -- bit 4        linkStatusOfnterfaceB       -- True: lié, False: non lié
        -- bit 3        linkStatusOfnterfaceA       -- True: lié, False: non lié
        -- bit 2        statusOfNetworkB            -- True: sain, False: défaillant
        -- bit 1        statusOfNetworkA            -- True: sain, False: défaillant
```

```
}

Nm_PublicKey ::= Unsigned64

Nm_ListOfPathStatus ::= CompactBooleanArray          -- True: sain, False: défaillant

Nm_DoaminInformation ::= SEQUENCE {
        NodeIdentifier              [0]  IMPLICIT  Nm_NodeIdentifier,
        NoOfInterfaces              [1]  IMPLICIT  Integer8,
        InterfaceID                 [2]  IMPLICIT  Unsigned8,
        localNodeTime               [3]  IMPLICIT  Tm_Time,
        diagInterval                [4]  IMPLICIT  BinaryTime2,
}

Nm_DomainStatus ::= BitString8 {
        -- bit 8        statusOfNetworkB              -- True: sain, False: défaillant
        -- bit 7        statusOfNetworkA              -- True: sain, False: défaillant
        -- bits 6 à 5   StatusOfTimeSynchronization   -- 00: non synchronisé
                                                      -- 01: synchronisé avec le maître de temps domaine
                                                      -- 10: synchronisé avec le maître de temps réseau
                                                      -- 11: synchronisé avec la source de temps externe
        -- bits 4 à 1   TimeGroup
}

Nm_NodeIdentifier ::= SEQUENCE {
        DomainNumber                [0]  IMPLICIT  Integer8,
        StationNumber               [1]  IMPLICIT  Integer8
}
```

4.4.7 Types généraux

4.4.7.1 Gn_KeyAttribute

```
Gn_KeyAttribute ::= CHOICE {
-- Lorsque ce type est spécifié, seuls les attributs clés de la classe référencée sont valides.
        numericID                   [0]  IMPLICIT  Gn_NumericID,
        name                        [1]  IMPLICIT  Gn_Name,
        listName                    [2]  IMPLICIT  Gn_Name,
        numericAddress              [4]  IMPLICIT  Gn_NumericAddress,
        symbolicAddress             [5]  IMPLICIT  Gn_SymbolicAddress
}
```

4.4.7.2 Gn_Name

```
Gn_Name ::= octetString
```

4.4.7.3 Gn_NumericAddress

```
Gn_NumericAddress ::= SEQUENCE {
        startAddress        [0]  IMPLICIT  Unsigned32,  -- adresse physique de l'emplacement de
                                                        départ
        length              [1]  IMPLICIT  Unsigned16  -- longueur en octets d'un bloc mémoire
}
```

4.4.7.4 Gn_NumericID

```
Gn_NumericID ::= Unsigned16          -- Les valeurs de ce paramètre sont uniques au sein d'un AP.
```

4.4.7.5 Gn_SymbolicAddress

```
Gn_SymbolicAddress ::= VisibleString
```

4.4.7.6 Gn_FullyNestedTypeDescription

```
Gn_FullyNestedTypeDescription ::= CHOICE {
        boolean                 [1]             Unsigned8,
        integer8                [2]             Unsigned8,
        integer16               [3]             Unsigned8,
        integer32               [4]             Unsigned8,
        unsigned8               [5]             Unsigned8,
        unsigned16              [6]             Unsigned8,
        unsigned32              [7]             Unsigned8,
        float32                 [8]             Unsigned8,
        float64                 [9]             Unsigned8,
        binaryDate              [10]            Unsigned8,
        timeOfDay               [11]            Unsigned8,
        timeDifference          [12]            Unsigned8,
        universalTime           [13]            Unsigned8,
        fieldbusTime            [14]            Unsigned8,
        time                    [15]            Unsigned8,
        bitstring8              [16]            Unsigned8,
        bitstring16             [17]            Unsigned8,
        bitstring32             [18]            Unsigned8,
        visiblestring1          [19]            Unsigned8,
        visiblestring2          [20]            Unsigned8,
        visiblestring4          [21]            Unsigned8,
        visiblestring8          [22]            Unsigned8,
        visiblestring16         [23]            Unsigned8,
        octetstring1            [24]            Unsigned8,
        octetstring2            [25]            Unsigned8,
        octetstring4            [26]            Unsigned8,
        octetstring8            [27]            Unsigned8,
        octetstring16           [28]            Unsigned8,
        bcd                     [29]            Unsigned8,
        iso10646char            [30]            Unsigned8,
        binarytime0             [31]            Unsigned8,
        binarytime1             [32]            Unsigned8,
        binarytime2             [33]            Unsigned8,
        binarytime3             [34]            Unsigned8,
        binarytime4             [35]            Unsigned8,
        binarytime5             [36]            Unsigned8,
        binarytime6             [37]            Unsigned8,
        binarytime7             [38]            Unsigned8,
        binarytime8             [39]            Unsigned8,
        binarytime9             [40]            Unsigned8,
        visiblestring           [41]            Unsigned8,
        octetstring             [42]            Unsigned8,
        bitstring               [43]            Unsigned8,
        compactBooleanArray     [44]            Unsigned8,
        compactBCDArray         [45]            Unsigned8,
        iso646string            [46]            Unsigned8,
        structure               [47]  IMPLICIT  SEQUENCE OF Gn_FullyNestedTypeDescription
}
```

4.5 Types de données

4.5.1 Notation du type Boolean

```
Boolean ::= BOOLEAN                     -- TRUE si la valeur est non nulle.
                                        -- FALSE si la valeur est nulle.
```

4.5.2 Notation du type Integer

```
Integer ::= INTEGER                     -- n'importe quel entier
Integer8 ::= INTEGER (-128..+127)       -- -27 <= i <= 27-1
Integer16 ::= INTEGER (-32768..+32767)  -- -215 <= i <= 215-1
Integer32 ::= INTEGER                   -- -231 <= i <= 231-1
```

4.5.3 Notation du type Unsigned

```
Unsigned ::= INTEGER                    -- n'importe quel entier naturel
Unsigned8 ::= INTEGER (0..255)          -- 0 <= i <= 28-1
Unsigned16 ::= INTEGER (0..65535)       -- 0 <= i <= 216-1
Unsigned32 ::= INTEGER                  -- 0 <= i <= 232-1
```

4.5.4 Notation du type Floating Point

```
Floating32 ::= BIT STRING SIZE (4)          -- CEI 60559 simple précision
Floating64 ::= BIT STRING SIZE ( 8)         -- CEI 60559 double précision
```

4.5.5 Notation du type BitString

```
BitString ::= BIT STRING                    -- Usage général
BitString4 ::= BIT STRING SIZE (4)          -- Chaîne binaire fixe de 4 bits
BitString8 ::= BIT STRING SIZE (8)          -- Chaîne binaire fixe de 8 bits
BitString16 ::= BIT STRING SIZE (16)        -- Chaîne binaire fixe de 16 bits
BitString32 ::= BIT STRING SIZE (32)        -- Chaîne binaire fixe de 32 bits
```

4.5.6 Notation du type octetString

```
octetString ::= OCTET STRING                -- Usage général
octetString2 ::= OCTET STRING SIZE (2)      -- Chaîne d'octets fixe de 2 octets
octetString4 ::= OCTET STRING SIZE (4)      -- Chaîne d'octets fixe de 4 octets
octetString6 ::= OCTET STRING SIZE (6)      -- Chaîne d'octets fixe de 6 octets
octetString7 ::= OCTET STRING SIZE (7)      -- Chaîne d'octets fixe de 7 octets
octetString8 ::= OCTET STRING SIZE (8)      -- Chaîne d'octets fixe de 8 octets
octetString16 ::= OCTET STRING SIZE (16)    -- Chaîne d'octets fixe de 16 octets
```

4.5.7 Notation du type VisibleString

```
VisibleString2 ::= VisibleString SIZE (2)   -- Chaîne visible fixe de 2 octets
VisibleString4 ::=VisibleString SIZE (4)    -- Chaîne visible fixe de 4 octets
VisibleString8 ::= VisibleString SIZE (8)   -- Chaîne visible fixe de 8 octets
VisibleString16 ::= VisibleString SIZE (16) -- Chaîne visible fixe de 16 octets
```

4.5.8 Notation du type UNICODEString

```
UNICODEString ::= UNICODEString             -- Jeu de codes de caractères de 16 bits défini dans l'ISO 10646.
```

4.5.9 Notation du type BinaryTime

```
BinaryTime0 ::= BIT STRING SIZE (16)        -- Résolution de 10 µs
BinaryTime1 ::= BIT STRING SIZE (16)        -- Résolution de 0,1 ms
BinaryTime2 ::= BIT STRING SIZE (16)        -- Résolution d'1 ms
BinaryTime3 ::= BIT STRING SIZE (16)        -- Résolution de 10 ms
BinaryTime4 ::= BIT STRING SIZE (16)        -- Résolution de 0,1 s
BinaryTime5 ::= BIT STRING SIZE (16)        -- Résolution d'1 s
BinaryTime6 ::= BIT STRING SIZE (32)        -- Résolution de 10 µs
BinaryTime7 ::= BIT STRING SIZE (32)        -- Résolution de 0,1 ms
BinaryTime8 ::= BIT STRING SIZE (32)        -- Résolution d'1 ms
BinaryTime9 ::= BIT STRING SIZE (32)        -- Résolution de 10 ms
```

4.5.10 Notation du type BCD

```
BCD ::= Unsigned8 (0..9)                    -- Les 4 bits inférieurs permettent d'exprimer une seule
                                            valeur BCD.
```

4.5.11 Notation du type CompactBooleanArray

```
CompactBooleanArray ::= BitString           -- Chaque bit de valeur 0 représente la valeur booléenne FALSE.
                                            -- Chaque bit de valeur 1 représente la valeur booléenne TRUE.
                                            -- Les bits non utilisés, le cas échéant, doivent être placés sur les
                                            bits 7 à 1 du dernier octet.
```

4.5.12 Notation du type CompactBCDArray

```
CompactBCDArray ::= octetString             -- Une valeur BCD est représentée par 4 bits; tout
                                            -- quartet non utilisé, le cas échéant, doit être placé sur les bits 4
                                            à 1 du dernier octet
                                            -- et défini sur 1111F.
```

5 Syntaxe de transfert

5.1 Vue d'ensemble du codage

Les unités PDU FAL codées doivent posséder un format uniforme. Les unités PDU FAL doivent se composer de deux parties principales, la partie "En-tête d'unité APDU" et la partie "Corps d'unité APDU", comme illustré à la Figure 1.

(1)	(1)	(1)	(n) --- octets
Champ FalArHeader	Champ Type	(InvokeID)	Paramètres spécifiques au service
<---------- En-tête d'unité APDU ---------->			<---------- Corps d'unité APDU ---------->

NOTE La présence du champ InvokeID dépend du type de l'unité APDU.

Figure 1 – Vue d'ensemble d'une unité APDU

Pour réaliser une unité APDU efficace tout en maintenant la flexibilité du codage, des règles de codage différentes sont utilisées pour les parties en-tête d'unité APDU et corps d'unité APDU.

NOTE Le service de couche de liaison de données propose un paramètre DLSDU qui définit la longueur de l'unité APDU. Les informations relatives à la longueur de l'unité APDU ne sont donc pas incluses dans l'unité APDU elle-même.

5.2 Codage de l'en-tête des unités APDU

La partie en-tête d'unité APDU est toujours présente dans toutes les unités APDU conformes à la présente norme. Elle se compose de trois champs: le champ FalArHeader, le champ Type et le champ InvokeID, qui est facultatif.

Ils sont décrits à la Figure 1.

5.2.1 Codage du champ FalArHeader

Toutes les unités PDU de couche FAL doivent posséder un en-tête d'unité PDU commun appelé FalArHeader. FalArHeader identifie la syntaxe abstraite, la syntaxe de transfert, ainsi que chacune des unités PDU. Le Tableau 2 définit la manière dont cet en-tête doit être utilisé.

Tableau 2 – Codage du champ FalArHeader

Position binaire de FalArHeader			Type d'unité PDU	Version de protocole
8 7	6 5 4	3 2 1		
01	001	000	ConfirmedSend-CommandPDU	Version 1
01	001	100	ConfirmedSend-ResponsePDU	Version 1
01	010	000	UnconfirmedSend-CommandPDU	Version 1

NOTE Tous les autres points de code sont réservés en vue de l'ajout de protocoles supplémentaires et de révisions futures.

5.2.2 Codage du champ Type

a) Le type de service d'une unité APDU est codé dans le champ Type; ce dernier correspond toujours au deuxième octet des unités APDU.

b) Tous les bits du champ Type permettent de coder le type de service.

1) Les types de service doivent être codés sur les bits 8 à 1 du champ Type, le bit 8 correspondant au bit de poids fort et le bit 1 au bit de poids faible. Le type de service doit être compris entre 0 (zéro) et 254 inclus.

2) La valeur 255 est réservée en vue d'extensions futures de la présente spécification.

3) Le type de service est spécifié dans la syntaxe abstraite sous la forme d'une valeur entière positive.

c) La Figure 2 illustre le codage du champ Type.

Figure 2 – Champ Type

5.2.3 Codage du champ InvokeID

Le champ InvokeID doit être présent s'il est indiqué dans la syntaxe abstraite. Dans le cas contraire, il ne doit pas être présent. Si ce champ est présent, il doit recevoir le paramètre InvokeID fourni par une primitive de service.

5.3 Codage du corps des unités APDU

5.3.1 Généralités

Les règles de codage de la couche FAL sont basées sur les termes et conventions définis dans l'ISO/CEI 8825-1. Le codage comprend trois composants, qui s'enchaînent dans l'ordre suivant:

composant octet d'identification;
composant octet(s) de longueur;
composant octet(s) de contenu.

5.3.2 Composant octet d'identification

Le composant octet d'identification doit coder l'étiquette définie dans la syntaxe abstraite de la couche FAL et se composer d'un seul octet.

Il comprend l'indicateur P/C et le champ Tag, comme illustré à la Figure 3.

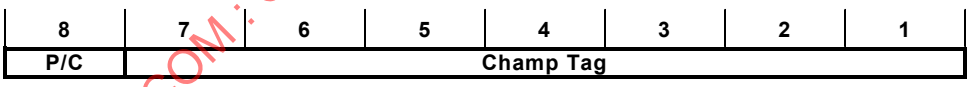


Figure 3 – Composant octet d'identification

L'indicateur P/C indique que le composant octet(s) de contenu correspond soit à un composant simple (types primitifs; par exemple, Integer8), soit à un composant structuré (types construits; par exemple, SEQUENCE ou SEQUENCE OF).

Si indicateur P/C = 0, le composant octet(s) de contenu correspond à un composant simple.
Si indicateur P/C = 1, le composant octet(s) de contenu correspond à un composant structuré.

Le champ Tag identifie la sémantique du composant octet(s) de contenu.

5.3.3 Composant octet(s) de longueur

Le composant octet(s) de longueur doit se composer d'un ou de trois octets.

a) Si la valeur du premier octet de longueur est différente de 255, aucun octet de longueur ne doit suivre et le premier octet doit contenir la valeur de l'octet de longueur défini ultérieurement.

b) Si la valeur du premier octet de longueur est 255, deux octets de longueur doivent suivre et ils doivent contenir la valeur des octets de longueur définis ultérieurement. Dans ce cas,

les informations de longueur du composant octet(s) de contenu doivent être représentées par les deux derniers octets du composant octet(s) de longueur, le bit de poids fort du deuxième des trois octets de longueur devant correspondre au bit de poids fort de la valeur de longueur et le bit de poids faible du troisième des trois octets de longueur devant correspondre au bit de poids faible de la valeur de longueur.

L'expéditeur doit avoir la possibilité de choisir entre le format à un octet et le format à trois octets. Par exemple, le format à trois octets peut être utilisé pour transmettre une valeur de longueur de un.

La signification du composant octet(s) de longueur dépend du type de valeur codé. Si le codage du composant octet(s) de contenu est primitif, le composant octet(s) de longueur doit contenir le même nombre d'octets que le composant octet(s) de contenu. Si le codage du composant octet(s) de contenu est construit, le composant octet(s) de longueur doit contenir le nombre des composants de premier niveau du composant octet(s) de contenu.

La Figure 4 et la Figure 5 présentent des exemples de codage du composant octet(s) de longueur.

8	7	6	5	4	3	2	1
(msb)	valeur de l'octet de longueur défini ci-dessus						(lsb)

Figure 4 – Octet de longueur (format à un octet)

1er octet	15	deuxième et troisième octets	1
11111111	(msb)	valeur des octets de longueur définis ci-dessus	(lsb)

Figure 5 – Octets de longueur (format à trois octets)

5.3.4 Composant octet(s) de contenu

Le composant octet(s) de contenu doit coder la valeur de donnée selon la règle de codage définie pour son type.

Le codage du composant octet(s) de contenu doit prendre l'une des deux formes suivantes: codage primitif ou codage construit.

a) Si le composant octet(s) de contenu contient un codage primitif, il représente un codage d'une seule valeur.

b) Si le composant octet(s) de contenu contient un codage construit, il représente un codage énuméré de plusieurs valeurs.

5.4 Règles de codage des types de données

5.4.1 Généralités

5.4.1.1 Boolean

Une valeur Boolean doit être codée comme suit:

a) Le composant octet d'identification et le composant octet(s) de longueur ne doivent pas être présents.

b) Le composant octet(s) de contenu se compose toujours d'un seul octet. Si la valeur Boolean est égale à FALSE, tous les bits de l'octet sont définis sur 0. Si la valeur Boolean est égale à TRUE, l'octet peut contenir n'importe quelle combinaison de bits à l'exception du codage de FALSE.

5.4.1.2 Integer

Une valeur Integer doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent si la taille de la valeur Integer est invariable. Un entier de taille invariable est créé en forçant la valeur possible. Le composant octet(s) de longueur doit être présent si la taille de la valeur Integer est variable.

c) Le composant octet(s) de contenu doit contenir le nombre binaire en complément à deux égal à la valeur Integer. Les huit bits de poids fort de la valeur Integer sont codés sur les bits 8 à 1 du premier octet, les huit bits suivants sur les bits 8 à 1 de l'octet suivant, et ainsi de suite. Si les valeurs d'un type Integer sont limitées aux nombres négatifs et naturels, le bit 8 du premier octet indique le signe de la valeur; si ces valeurs sont limitées exclusivement aux nombres naturels, aucun bit de signe n'est nécessaire.

5.4.1.3 Valeur Unsigned

Une valeur Unsigned doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent si la taille de la valeur Unsigned est invariable. La longueur d'une valeur Unsigned de taille invariable dépend de la plage spécifiée de la valeur. Le composant octet(s) de longueur doit être présent si la taille de la valeur Unsigned est variable.

c) Le composant octet(s) de contenu doit correspondre à un nombre binaire égal à la valeur Unsigned; il doit se composer des bits 8 à 1 du premier octet, suivis des bits 8 à 1 du deuxième octet, et ainsi de suite jusqu'au dernier octet du composant octet(s) de contenu inclus.

5.4.1.4 Floating Point

Une valeur Floating Point doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu doit contenir des valeurs Floating Point définies conformément à la CEI 60559. Le signe est codé au moyen du bit 8 du premier octet. Il est suivi de l'exposant à partir du bit 7 du premier octet, puis de la mantisse à partir du bit 7 du deuxième octet pour le type Floating32 ou du bit 4 du deuxième octet pour le type Floating64.

5.4.1.5 BitString

Une valeur BitString doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent si la taille de la valeur BitString est invariable. Une chaîne binaire de taille invariable est créée par l'application d'une contrainte de taille contenant une seule valeur au type BitString. Le composant octet(s) de longueur doit être présent si la taille de la valeur BitString est variable.

c) Le composant octet(s) de contenu comporte autant d'octets que nécessaire pour recevoir l'ensemble des bits de la valeur réelle: N_octets = (N_Bits-1) div 8 + 1. La valeur BitString qui commence par le premier bit et va jusqu'au dernier bit doit être placée sur les bits 8 à 1 du premier octet, suivis des bits 8 à 1 du deuxième octet, et ainsi de suite. Si le nombre de bits n'est pas un multiple de 8, on trouve des bits non utilisés, qui sont situés sur les bits de poids faible du dernier octet. La valeur des bits non utilisés peut être zéro (0) ou un (1) et n'avoir aucune signification.

5.4.1.6 OctetString

Une valeur octetString doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent si la taille de la valeur octetString est invariable. Une chaîne d'octets de taille invariable est créée par l'application d'une contrainte de taille contenant une seule valeur au type octetString. Le composant octet(s) de longueur doit être présent si la taille de la valeur octetString est variable.

c) La valeur du composant octet(s) de contenu doit être égale à celle des octets de la valeur de donnée.

5.4.1.7 VisibleString

Une valeur VisibleString doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent si la taille de la valeur VisibleString est invariable. Une chaîne visible de taille invariable est créée par l'application d'une contrainte de taille contenant une seule valeur au type VisibleString. Le composant octet(s) de longueur doit être présent si la taille de la valeur VisibleString est variable.

c) La valeur du composant octet(s) de contenu doit être égale à celle des octets de la valeur de donnée.

5.4.1.8 UNICODEString (chaîne ISO 10646)

Une valeur UNICODEString doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur doit indiquer le nombre d'octets du composant octet(s) de contenu sous la forme d'un nombre binaire.

c) Chaque caractère ISO 10646 doit être placé sur deux octets du composant octet(s) de contenu, l'octet de poids fort étant placé sur le premier octet et l'octet de poids faible sur l'octet suivant, et le bit de poids fort d'un octet de la valeur de donnée étant aligné avec le bit de poids fort d'un octet du composant octet(s) de contenu.

5.4.1.9 BinaryTime

Une valeur BinaryTime doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu doit correspondre à un nombre binaire égal à la valeur BinaryTime; il doit se composer des bits 8 à 1 du premier octet, suivis des bits 8 à 1 du deuxième octet, et ainsi de suite jusqu'au dernier octet du composant octet(s) de contenu inclus.

5.4.1.10 BCD

a) Une valeur BCD doit être codée de la même manière qu'une valeur Unsigned8.

b) Une valeur BCD doit être placée sur les bits 4 à 1 de l'octet de contenu d'une valeur Unsigned8. La valeur des bits 8 à 5 doit être zéro (0).

5.4.1.11 CompactBooleanArray

Une valeur CompactBooleanArray doit être codée de la même manière qu'une valeur BitString.

5.4.1.12 CompactBCDArray

a) Une valeur CompactBCDArray doit être codée de la même manière qu'un type primitif.

b) Le composant octet d'identification ne doit pas être présent.

c) Le composant octet(s) de longueur doit indiquer le nombre d'octets de la matrice sous la forme d'un nombre binaire.

d) Si le nombre des valeurs BCD est zéro, aucun octet ne doit suivre et le composant octet(s) de longueur doit être défini sur zéro.

e) La première valeur BCD doit être placée sous la forme d'un nombre binaire sur les bits 8 à 5 du premier composant octet(s) de contenu et la deuxième sur les bits 4 à 1 du premier composant octet(s) de contenu. L'opération sera renouvelée pour les valeurs BCD et les octets de contenu restants jusqu'au dernier octet de contenu inclus. La valeur des éventuels bits non utilisés du dernier octet de contenu doit être définie sur 1.

5.4.1.13 SEQUENCE

Une valeur de type SEQUENCE doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur doit être présent et spécifier le nombre des composants de premier niveau du composant octet(s) de contenu. Toutefois, cette longueur ne doit pas être codée pour le premier mot-clé SEQUENCE de FalArPDU.

c) Le composant octet(s) de contenu doit comporter les codages de tous les types d'élément, dans l'ordre dans lequel ils sont spécifiés dans la description ASN.1 du type SEQUENCE.

5.4.1.14 SEQUENCE OF

Une valeur de type SEQUENCE OF doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur doit être présent et spécifier le nombre des composants de premier niveau du composant octet(s) de contenu.

c) Le composant octet(s) de contenu doit comporter les codages de tous les types d'élément, dans l'ordre dans lequel ils sont spécifiés dans la description ASN.1 du type SEQUENCE OF.

5.4.1.15 CHOICE

Une valeur de type CHOICE doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu doit comporter le codage du type choisi dans la liste des types de remplacement.

5.4.1.16 NULL

Une valeur de type NULL doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu ne doit pas être présent.

5.4.1.17 Tagged

Une valeur de type Tagged doit être codée comme suit:

a) Le composant octet d'identification ne doit être présent que si le type Tagged fait partie d'une liste de types de remplacement dans une construction CHOICE.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu doit comporter le codage du type étiqueté.

5.4.1.18 IMPLICIT

Une valeur de type IMPLICIT doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu doit comporter le codage du type référencé par la construction IMPLICIT, sauf lorsque le type référencé est un type SEQUENCE. Dans ce cas, le composant octet(s) de contenu se compose uniquement du composant octet(s) de contenu du type SEQUENCE référencé et le composant octet(s) de longueur de ce type SEQUENCE ne doit pas être présent.

5.4.1.19 OPTIONAL et DEFAULT

Une valeur de type OPTIONAL ou DEFAULT doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur doit être présent. Si aucune valeur de ce type n'est présente, le composant octet(s) de longueur contient la valeur 0.

c) Le composant octet(s) de contenu doit comporter le codage du type référencé si une valeur de ce type est présente; dans le cas contraire, aucun octet de contenu n'est présent.

5.4.1.20 ANY

Le type ANY est utilisé pour définir des types complexes, dont la structure est décrite de manière informelle plutôt que dans ASN.1.

Une valeur de type ANY doit être codée comme suit:

a) Le composant octet d'identification ne doit pas être présent.

b) Le composant octet(s) de longueur ne doit pas être présent.

c) Le composant octet(s) de contenu doit comporter le codage de tous les types IMPLICIT constituant le type ANY.

6 Structure des diagrammes d'états de protocole de la couche FAL

Le présent paragraphe spécifie les machines de protocole de la couche FAL et l'interface qui les sépare.

NOTE Les diagrammes d'états spécifiés dans le présent article et les machines ARPM définies dans les articles suivants définissent uniquement les événements relatifs au protocole pour chacun. Le traitement des autres événements s'effectue localement.

Le comportement de la couche FAL est décrit par trois machines de protocole intégrées. Les trois types de machine de protocole sont les suivants: les machines de protocole de service FAL (FSPM), les machines de protocole de relations d'applications (ARPM) et les machines de protocole de mapping de la couche de liaison de données (Data-link layer Mapping Protocol Machine, DMPM). Des machines de protocole spécifiques sont définies pour les différents types d'AREP. Les relations et les primitives échangées entre ces machines de protocole sont représentées à la Figure 6.

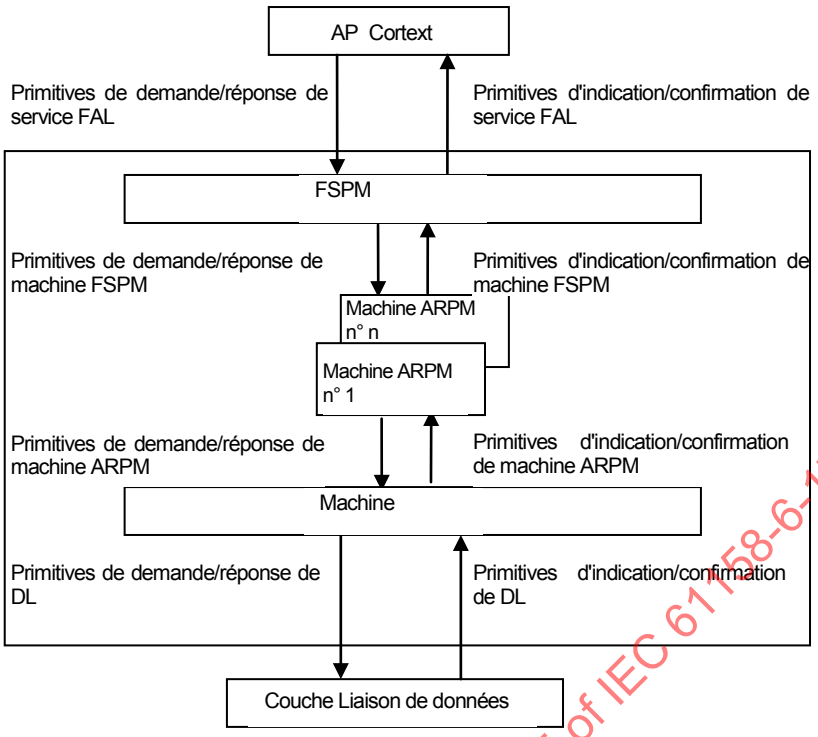


Figure 6 – Relations entre les machines de protocole et les couches adjacentes

La machine FSPM est chargée des activités suivantes:

a) accepter les primitives de service provenant de l'utilisateur de service FAL et les convertir en primitives de couche FAL internes;

b) sélectionner un diagramme d'états ARPM approprié en fonction du paramètre d'identificateur d'AREP fourni par l'entité AP-Context et envoyer les primitives de couche FAL internes à la machine ARPM choisie;

c) accepter les primitives de couche FAL internes provenant de la machine ARPM et les convertir en primitives de service à l'intention de l'entité AP-Context;

d) remettre les primitives de service FAL à l'entité AP-Context en fonction du paramètre d'identificateur d'AREP associé aux primitives.

La machine ARPM est chargée des activités suivantes:

a) accepter les primitives de couche FAL internes provenant de la machine FSPM, et créer et envoyer d'autres primitives de couche FAL internes soit à la machine FSPM, soit à la machine DMPM, en fonction du type de primitive et d'AREP;

b) accepter les primitives de couche FAL internes provenant de la machine DMPM et les envoyer à la machine FSPM sous une forme convertie à l'intention de la machine FSPM;

c) si les primitives concernent le service Establish (établissement) ou Abort (interruption), elle doit essayer d'établir ou de libérer l'AR spécifiée.

La machine DMPM décrit le mapping entre la couche FAL et la couche DLL. Elle est commune à tous les types d'AREP et ne présente pas de changements d'état. La machine DMPM est chargée des activités suivantes:

a) accepter les primitives de couche FAL internes provenant de la machine ARPM, préparer les primitives de service de couche DLL et les envoyer à la couche DLL;

b) recevoir les primitives d'indication ou de confirmation provenant de la couche DLL et les envoyer à la machine ARPM sous une forme convertie à l'intention de la machine ARPM.

7 Diagramme d'états de contexte d'application

Aucun diagramme d'états de contexte d'application n'est défini pour ce protocole.

8 Machines de protocole de service FAL (FSPM)

8.1 Généralités

Les différentes machines de protocole de service FAL sont les suivantes:

- la machine de protocole d'élément ASE de variable (Variable ASE Protocol Machine, VARM);
- la machine de protocole d'élément ASE d'événement (Event ASE Protocol Machine, EVTM);
- la machine de protocole d'élément ASE de région de charge (Load Region ASE Protocol Machine, LDRM);
- la machine de protocole d'élément ASE d'invocation de fonctions (Function Invocation ASE Protocol Machine, FNIM);
- la machine de protocole d'élément ASE de temps (Time ASE Protocol Machine, TIMM);
- la machine de protocole d'élément ASE de gestion de réseau (Network Management ASE Protocol Machine, NWMM).

8.2 Paramètres communs des primitives

De nombreux services possèdent les paramètres ci-dessous. Au lieu de les définir avec chaque service, on établit les définitions communes ci-dessous.

AREP
Ce paramètre contient des informations suffisantes pour identifier l'AREP à utiliser pour transmettre le service. Ce paramètre peut utiliser un attribut clé de l'AREP pour identifier la relation d'applications. Lorsqu'un AREP prend en charge plusieurs contextes (établis à l'aide du service Initiate (lancement)) en même temps, le paramètre AREP est étendu pour identifier le contexte ainsi que l'AREP.

InvokeID
Ce paramètre identifie cette invocation du service. Il permet d'associer une demande de service à la réponse correspondante. Deux invocations de services en cours ne peuvent donc pas être identifiées par la même valeur InvokeID.

Error Info
Ce paramètre fournit des informations d'erreur pour les erreurs de service. Il est renvoyé dans les primitives de service confirmé et les primitives de réponse.

8.3 Machine de protocole d'élément ASE de variable (VARM)

8.3.1 Définition des primitives

8.3.1.1 Primitives échangées

Le Tableau 3 répertorie les primitives de service échangées entre l'utilisateur FAL et la machine VARM, avec les paramètres associés.

Tableau 3 – Primitives échangées entre l'utilisateur FAL et la machine VARM

Nom de la primitive	Source	Paramètres associés	Fonctions
Read.req	Utilisateur FAL	VariableSpecifier	Cette primitive permet de lire des valeurs au niveau de variables distantes.
Write.req	Utilisateur FAL	VariableSpecifier	Cette primitive permet d'écrire des valeurs au

Nom de la primitive	Source	Paramètres associés	Fonctions
			niveau de variables distantes.
InfReport.req	Utilisateur FAL	VariableSpecifier, Value, RemoteArep	Cette primitive permet de publier des variables.
Read.rsp	Utilisateur FAL	VariableSpecifier, Value, ErrorInfo	Cette primitive permet de transmettre les valeurs de variables demandées.
Write.rsp	Utilisateur FAL	VariableSpecifier, ErrorInfo	Cette primitive permet de signaler le résultat de l'écriture demandée.
Read.ind	VARM	VariableSpecifier	Cette primitive permet de transmettre une demande de lecture.
Write.ind	VARM	VariableSpecifier Value	Cette primitive permet de transmettre une demande d'écriture.
InfReport.ind	VARM	VariableSpecifier, Value	Cette primitive permet de signaler les valeurs de variables publiées.
Read.cnf	VARM	VariableSpecifier, Value ErrorInfo	Cette primitive permet de transmettre les valeurs de variables demandées, ainsi que le résultat de la lecture.
Write.cnf	VARM	VariableSpecifier, ErrorInfo	Cette primitive permet de signaler le résultat de l'écriture demandée.

8.3.1.2 Paramètres des primitives

Le Tableau 4 énumère les paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine VARM.

Tableau 4 – Paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine VARM

Nom du paramètre	Description
VariableSpecifier	Ce paramètre spécifie une variable ou une liste de variables.
RemoteArep	Ce paramètre spécifie un AREP distant auquel l'unité APDU doit être transférée.
Value	Ce paramètre contient la valeur de variable à lire/écrire.
ErrorInfo	Ce paramètre fournit des informations d'erreur pour les erreurs de service.

8.3.2 Diagramme d'états

8.3.2.1 Généralités

Le diagramme d'états de la machine VARM dispose d'un seul état possible: ACTIVE (actif).

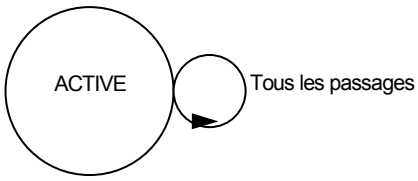


Figure 7 – Diagramme de passages d'état de la machine VARM

8.3.2.2 Tables d'états

La Figure 7, le Tableau 5 et le Tableau 6 décrivent le diagramme d'états de la machine VARM.

Tableau 5 – Table d'états de la machine VARM: passages expéditeur

N°	État actuel	Événement ou condition => action	État suivant
S1	ACTIVE	Read.req => ArepID := GetArep(VariableSpecifier) SelectArep(ArepID, “PTC-AR”),	ACTIVE

N°	État actuel	Événement ou condition => action	État suivant

		CS_req{
		user_data := Read-RequestPDU
		}	

S2	ACTIVE	Write.req
		=>
		ArepID := GetArep(VariableSpecifier)
		SelectArep(ArepID, “PTC-AR”),
		CS_req{
		user_data := Write-RequestPDU
		}
			ACTIVE

S3	ACTIVE	InfReport.req
		=>
		SelectArep(RemoteArep, “MSU-AR”),
		UCS_req{
		user_data := InformationReport-RequestPDU
		}
			ACTIVE

S4	ACTIVE	Read.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Read-ResponsePDU
		}
			ACTIVE

S5	ACTIVE	Write.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		user_data := Write-ResponsePDU
		}
			ACTIVE

Tableau 6 – Table d'états de la machine VARM: passages destinataire

N°	État actuel	Événement ou condition => action	État suivant

R1	ACTIVE	CS_ind
		&& PDU_Type = Read_RequestPDU
		=>
		Read.ind{
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	CS_ind
		&& PDU_Type = Write_RequestPDU
		=>
		Write.ind{
		ArepID := arep_id
		Data := user_data,
		}
			ACTIVE

R3	ACTIVE	CS_ind
		&& PDU_Type = Read_ResponsePDU
		&& GetErrorInfo() = “success”
		=>
		Read.cnf(+){
		Data := user_data
		}
			ACTIVE

R4	ACTIVE	CS_ind
		&& PDU_Type = Read_ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Read.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R5	ACTIVE	CS_ind
		&& PDU_Type =Write_ResponsePDU
		&& GetErrorInfo() = “success”
		=>
		Write.cnf(+){
		Data := user_data
		}
			ACTIVE

R6	ACTIVE	CS_ind
		&& PDU_Type = Write_ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Write.cnf(-){
		ErrorInfo := GetErrorInfo()
		}
			ACTIVE

R7	ACTIVE	UCS_ind
		&& PDU_Type = InformationReport-RequestPDU
		=>
		InfReport.ind{
		Data := user_data
		}
			ACTIVE

R8	ACTIVE	CS_cnf && Status = “success” => (aucune action)	ACTIVE

R9	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Read”
		=>
		Read.cnf(-){
		ErrorInfo := Status
		}
			ACTIVE

R10	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Write”
		=>
		Write.cnf(-){
		ErrorInfo := Status
		}
			ACTIVE

8.3.2.3 Fonctions

Le Tableau 7 énumère les fonctions utilisées par la machine VARM, ainsi que les arguments et les descriptions correspondants.

Tableau 7 – Fonctions utilisées par la machine VARM

Nom de la fonction	Paramètre	Description
SelectArep	ArepID, ARtype	Recherche l'entrée AREP spécifiée par ArepID and ARtype
GetArep	VariableSpecifier	Recherche ArepID en fonction de la valeur VariableSpecifier spécifiée
GetErrorInfo		Obtient des informations d'erreur de l'unité APDU
GetService	InvokeID	Obtient le nom de service de InvokeID

8.4 Machine de protocole d'élément ASE d'événement (EVTM)

8.4.1 Définition des primitives

8.4.1.1 Primitives échangées

Le Tableau 8 répertorie les primitives de service échangées entre l'utilisateur FAL et la machine EVTM, avec les paramètres associés.

Tableau 8 – Primitives échangées entre l'utilisateur FAL et la machine EVTM

Nom de la primitive	Source	Paramètres associés	Fonctions
Notification.req	Utilisateur FAL	AREP NotifierID Sequence Number ListOfEventMessages	Cette primitive permet de demander la publication de messages d'événement.
EventRecovery.req	Utilisateur FAL	AREP NotifierID SequenceNumber	Cette primitive permet de demander la retransmission de la notification d'événement.
Notification.ind	EVTM	AREP NotifierID SequenceNumber List of Event Messages	Cette primitive permet d'informer la notification d'événement.
EventRecovery.ind	EVTM	AREP NotifierID SequenceNumber	Cette primitive permet d'informer la demande de retransmission de la notification d'événement.

8.4.1.2 Paramètres des primitives

Le Tableau 9 énumère les paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine EVTM.

Tableau 9 – Paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine EVTM

Nom du paramètre	Description
NotifierID	Ce paramètre conditionnel identifie le notificateur à l'origine de la notification d'événement. Il est présent si plusieurs notificateurs sont définis pour l'AP.
SequenceNumber	Ce paramètre facultatif correspond au numéro de séquence de la notification d'événement. Il peut être utilisé dans le cadre de la récupération de notification.
NotificationTime	Ce paramètre facultatif correspond à l'heure de la notification d'événement.
ListOfEventMessages	Ce paramètre contient la liste des messages d'événement à signaler. Il peut contenir des messages provenant d'un ou de plusieurs objets d'événement, chaque objet contenant le même jeu de paramètres.

8.4.2 Diagramme d'états

8.4.2.1 Généralités

Le diagramme d'états de la machine EVTM dispose d'un seul état possible: ACTIVE (actif).

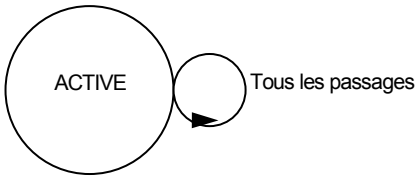


Figure 8 – Diagramme de passages d'état de la machine EVTM

8.4.2.2 Tables d'états

La Figure 8, le Tableau 10 et le Tableau 11 décrivent le diagramme d'états de la machine EVTM.

Tableau 10 – Table d'états de la machine EVTM: passages expéditeur

N°	Etat actuel	Evénement ou condition => action	Etat suivant

S1	ACTIVE	Notification.req
		=>
		SelectArep(RemoteArep, "MTU-AR"),
		UCS_req{
		user_data := Event-NotificationPDU
		}
			ACTIVE

S2	ACTIVE	EventRecovery.req
		=>
		SelectArep(RemoteArep, "PTU-AR"),
		UCS_req{
		arep := SelectArep(CalledAREP, "PTU-AR"),
		user_data := EventRecovery-RequestPDU
		}
			ACTIVE

Tableau 11 – Table d'états de la machine EVTM: passages destinataire

N°	Etat actuel	Evénement ou condition => action	Etat suivant

R1	ACTIVE	UCS_ind
		&& PDU_Type = Event_NotifiationPDU
		=>
		Notification.ind{
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	UCS_ind
		&& PDU_Type = EventRecovery-RequestPDU
		=>
		EventRecovery.ind {
		Data := user_data
		}
			ACTIVE

8.4.2.3 Fonctions

Le Tableau 12 présente la fonction utilisée par la machine EVTM, ainsi que les arguments et la description correspondante.

Tableau 12 – Fonction utilisée par la machine EVTM

Nom de la fonction	Paramètre	Description
SelectArep	ArepID, ARtype	Recherche l'entrée AREP spécifiée par ArepID and ARtype

8.5 Machine de protocole d'élément ASE de région de charge (LDRM)

8.5.1 Définition des primitives

8.5.1.1 Primitives échangées

Le Tableau 13 répertorie les primitives de service échangées entre l'utilisateur FAL et la machine LDRM, avec les paramètres associés.

Tableau 13 – Primitives échangées entre l'utilisateur FAL et la machine LDRM

Nom de la primitive	Source	Paramètres associés	Fonctions
Download.req	Utilisateur FAL	AREP InvokeID LoadRegion LoadData	Cette primitive permet de demander à ce que des données soient téléchargées dans la région.
Upload.req	Utilisateur FAL	AREP InvokeID LoadRegion	Cette primitive permet de demander à ce que des données soient chargées depuis la région.
Download.rsp	Utilisateur FAL	AREP InvokeID Error Info	Cette primitive permet de signaler le résultat du téléchargement demandé.
Upload.rsp	Utilisateur FAL	AREP InvokeID LoadData ErrorInfo	Cette primitive permet de transmettre les données à charger.
Download.ind	LDRM	AREP InvokeID LoadRegion LoadData	Cette primitive permet de transmettre les données téléchargées.
Upload.ind	LDRM	AREP InvokeID Load region	Cette primitive permet de transmettre une demande de chargement.
Download.cnf	LDRM	AREP InvokeID ErrorInfo	Cette primitive permet de transmettre un résultat de téléchargement.
Upload.cnf	LDRM	AREP InvokeID LoadData ErrorInfo	Cette primitive permet de transmettre les données chargées.

8.5.1.2 Paramètres des primitives

Le Tableau 14 énumère les paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine LDRM.

Tableau 14 – Paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine LDRM

Nom du paramètre	Description
LoadRegion	Ce paramètre spécifie la région à partir de laquelle ou vers laquelle l'image doit être chargée.
LoadData	Ce paramètre contient les données à charger.
ErrorInfo	Ce paramètre fournit des informations d'erreur pour les erreurs de service.

8.5.2 Diagramme d'états

8.5.2.1 Généralités

Le diagramme d'états de la machine LDRM dispose d'un seul état possible: ACTIVE (actif).

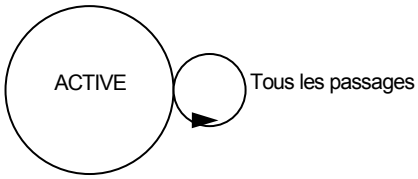


Figure 9 – Diagramme de passages d'état de la machine LDRM

8.5.2.2 Tables d'états

La Figure 9, le Tableau 15 et le Tableau 16 décrivent le diagramme d'états de la machine LDRM.

Tableau 15 – Table d'états de la machine LDRM: passages expéditeur

N°	État actuel	Événement ou condition => action	État suivant

S1	ACTIVE	Download.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := DownLoad-RequestPDU
		}
			ACTIVE

S2	ACTIVE	Upload.req
		=>
		SelectArep(RemoteArep, “PTC-AR”),
		CS_req{
		user_data := UpLoad-RequestPDU
		}
			ACTIVE

S3	ACTIVE	Download.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		arep := SelectArep(CallingAREP, “PTC-AR”),
		user_data := DownLoad-ResponsePDU
		}
			ACTIVE

S4	ACTIVE	Upload.rsp
		=>
		SelectArep(ArepID, “PTC-AR”),
		CS_rsp{
		arep := SelectArep(CallingAREP, “PTC-AR”),
		user_data := UpLoad-ResponsePDU
		}
			ACTIVE

Tableau 16 – Table d'états de la machine LDRM: passages destinataire

N°	État actuel	Événement ou condition => action	État suivant

R1	ACTIVE	CS_ind
		&& PDU_Type = DownLoad-RequestPDU
		=>
		Download.ind {
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	CS_ind
		&& PDU_Type = UpLoad-RequestPDU
		=>
		Upload.ind {
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R3	ACTIVE	CS_ind
		&& PDU_Type = DownLoad-ResponsePDU
		=>
		Download.cnf(+) {
		Data := user_data
		}
			ACTIVE

R4	ACTIVE	CS_ind
		&& PDU_Type = UpLoad-ResponsePDU
		=>
		Upload.cnf(+) {
		Data := user_data
		}
			ACTIVE

R5	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Download”
		=>
		Download.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

R6	ACTIVE	CS_cnf
		&& Status <> “success”
		&& GetService(InvokeID) = “Upload”
		=>
		Upload.cnf(-) {
		ErrorInfo := Status
		}
			ACTIVE

8.5.2.3 Fonctions

Le Tableau 17 énumère les fonctions utilisées par la machine LDRM, ainsi que les arguments et les descriptions correspondants.

Tableau 17 – Fonctions utilisées par la machine LDRM

Nom de la fonction	Paramètre	Description
SelectArep	ArepID, ARtype	Recherche l'entrée AREP spécifiée par ArepID and ARtype
GetErrorInfo		Obtient des informations d'erreur de l'unité APDU
GetService	InvokeID	Obtient le nom de service de InvokeID

8.6 Machine de protocole d'élément ASE d'invocation de fonctions (FNIM)

8.6.1 Définition des primitives

8.6.1.1 Primitives échangées

Le Tableau 18 répertorie les primitives de service échangées entre l'utilisateur FAL et la machine FNIM, avec les paramètres associés.

Tableau 18 – Primitives échangées entre l'utilisateur FAL et la machine FNIM

Nom de la primitive	Source	Paramètres associés	Fonctions
Start.req	Utilisateur FAL	AREP InvokeID FunctionID	Cette primitive permet de demander le démarrage de la fonction.
Stop.req	Utilisateur FAL	AREP InvokeID FunctionID	Cette primitive permet de demander l'arrêt de la fonction.
Resume.req	Utilisateur FAL	AREP InvokeID FunctionID	Cette primitive permet de demander la reprise de la fonction.
Start.rsp	Utilisateur FAL	AREP InvokeID Error Info	Cette primitive permet de signaler le résultat du démarrage demandé.
Stop.rsp	Utilisateur FAL	AREP InvokeID Error Info	Cette primitive permet de signaler le résultat de l'arrêt demandé.
Resume.rsp	Utilisateur FAL	AREP InvokeID Error Info	Cette primitive permet de signaler le résultat de la reprise demandée.
Start.ind	FNIM	AREP InvokeID FunctionID	Cette primitive permet de transmettre une demande de démarrage.
Stop.ind	FNIM	AREP InvokeID FunctionID	Cette primitive permet de transmettre une demande d'arrêt.
Resume.ind	FNIM	AREP InvokeID FunctionID	Cette primitive permet de transmettre une demande de reprise.
Start.cnf	FNIM	AREP InvokeID Error Info	Cette primitive permet de transmettre un résultat de démarrage.
Stop.cnf	FNIM	AREP InvokeID Error Info	Cette primitive permet de transmettre un résultat d'arrêt.
Resume.cnf	FNIM	AREP InvokeID Error Info	Cette primitive permet de transmettre un résultat de reprise.

8.6.1.2 Paramètres des primitives

Le Tableau 19 présente le paramètre utilisé avec les primitives échangées entre l'utilisateur FAL et la machine FNIM.

Tableau 19 – Paramètres utilisés avec les primitives échangées entre l'utilisateur FAL et la machine FNIM

Nom du paramètre	Description
FunctionID	Ce paramètre spécifie un des attributs clés de l'objet d'invocation de fonctions

8.6.2 Diagramme d'états

8.6.2.1 Généralités

Le diagramme d'états de la machine FNIM dispose d'un seul état possible: ACTIVE (actif).

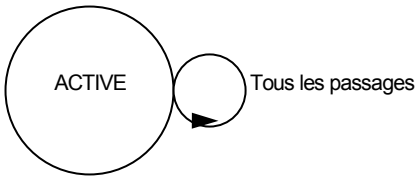


Figure 10 – Diagramme de passages d'état de la machine FNIM