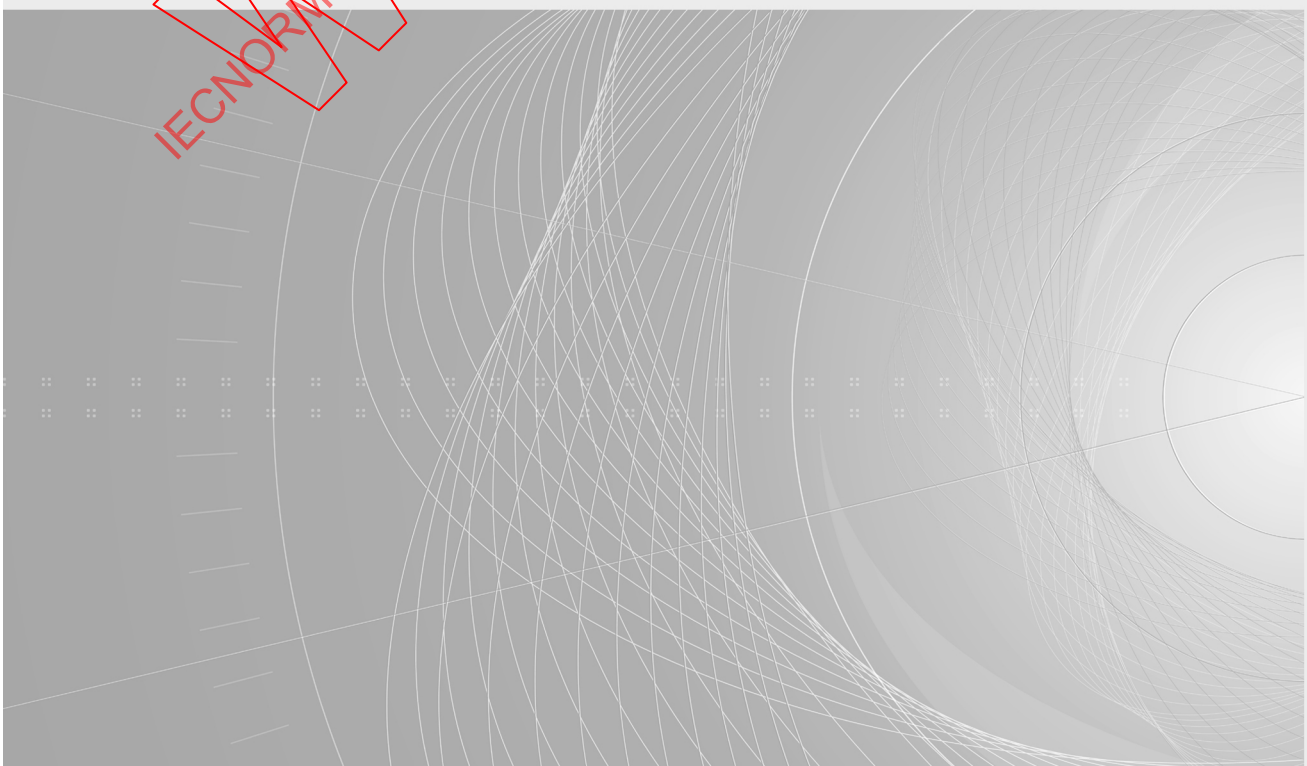
IEC 61158-6-9

Edition 2.0 2010-08

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-9: Application layer protocol specification – Type 9 elements



IEC 61158-6-9:2010(E)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: 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.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC 61158-6-9

Edition 2.0 2010-08

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-9: Application layer protocol specification – Type 9 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XD

ICS 25.04.40; 35.100.70; 35.110

ISBN 978-2-88912-127-4

CONTENTS

FOREWORD 5
INTRODUCTION 7
1 Scope 8
1.1 General 8
1.2 Specifications 8
1.3 Conformance 9
2 Normative references 9
3 Terms, definitions, symbols, abbreviations and conventions 9
3.1 Terms and definitions from other ISO/IEC standards 9
3.2 IEC/TR 61158-1 terms 10
3.3 Abbreviations and symbols 14
3.4 Conventions 15
3.5 Conventions used in state machines 15
4 Abstract syntax 16
4.1 FAL-AR PDU abstract syntax 16
4.2 Abstract syntax of PDUBody 19
4.3 Type definitions for ASEs 22
4.4 Abstract syntax of data types 27
5 Transfer syntax 28
5.1.1 General 28
5.1.2 Coding rules 28
5.1.3 Structure of the identification information 29
6 Structure of FAL protocol state machines 38
7 AP-Context state machines 40
7.1 VCR PM structure 40
7.2 VCR PM state machine 40
8 FAL service protocol machine (FSPM) 52
8.1 General 52
8.2 FSPM state tables 52
8.3 Functions used by FSPM 55
8.4 Parameters of FSPM/ARPM primitives 55
9 Application relationship protocol machines (ARPMs) 55
9.1 AREP mapping to data-link layer 55
9.2 Application relationship protocol machines (ARPMs) 65
9.3 AREP state machine primitive definitions 81
9.4 AREP state machine functions 83
10 DLL mapping protocol machine (DMPM) 84
10.1 DMPM States 84
10.2 DMPM state table 84
10.3 Primitives exchanged between data-link layer and DMPM 91
10.4 Functions used by DMPM 94
Bibliography 96

Figure 1 – Insertion of identification information in the FMS PDU 28
Figure 2 – Identification 29

Figure 3 – Coding with identification 30
Figure 4 – Coding without identification 30
Figure 5 – Representation of the value true 30
Figure 6 – Representation of the value false 31
Figure 7 – Coding of data of data type Integer16 31
Figure 8 – Coding of data of data type Unsigned16 32
Figure 9 – Coding of data of data type Floating Point 32
Figure 10 – Coding of data of data type Visible String 33
Figure 11 – Coding of data of data type Octet String 33
Figure 12 – Coding of data of type Date 34
Figure 13 – Coding of data of data type Time-of-day 35
Figure 14 – Coding of data of data type Time-difference 35
Figure 15 – Coding of data of data type Bit String 36
Figure 16 – Coding of data of data type Time-value 36
Figure 17 – Coding of data of user data definitions with identifier 37
Figure 18 – Coding of data of user data definitions without identifier 37
Figure 19 – Coding of ID info for a SEQUENCE 37
Figure 20 – Relationships among protocol machines and adjacent layers 39
Figure 21 – Relationships among protocol machines and adjacent layers 40
Figure 22 – VCR state machine 41
Figure 23 – State transition diagram of FSPM 52
Figure 24 – State transition diagram of the QUU ARPM 66
Figure 25 – State transition diagram of QUB ARPM 68
Figure 26 – State transition diagram of the BNU ARPM 76
Figure 27 – State transition diagram of DMPM 84

Table 1 – Conventions used for state machines 15
Table 2 – Coding for Date type 34
Table 3 – AP-VCR state machine transactions 42
Table 4 – Primitives issued by FAL-User to VCR PM 50
Table 5 – Primitives issued by VCR PM to FAL-User 51
Table 6 – Primitives issued by VCR PM to FSPM 51
Table 7 – Primitives issued by FSPM to VCR PM 52
Table 8 – FSPM state table – sender transactions 53
Table 9 – FSPM state table – receiver transactions 54
Table 10 – Function SelectArep() 55
Table 11 – Parameters used with primitives exchanged between FSPM and ARPM 55
Table 12 – QUU ARPM states 66
Table 13 – QUU ARPM state table – sender transactions 66
Table 14 – QUU ARPM state table – receiver transactions 67
Table 15 – QUB ARPM states 68
Table 16 – QUB ARPM state table – sender transactions 69
Table 17 – QUB ARPM state table – receiver transactions 70

Table 18 – BNU ARPM states 76
Table 19 – BNU ARPM state table – sender transactions 77
Table 20 – BNU ARPM state table – receiver transactions 78
Table 21 – Primitives issued from ARPM to DMPM 81
Table 22 – Primitives issued by DMPM to ARPM 81
Table 23 – Parameters used with primitives exchanged between ARPM and DMPM 82
Table 24 – Function GetArepId() 83
Table 25 – Function BuildFAS-PDU 83
Table 26 – Function FAS_Pdu_Type 83
Table 27 – Function AbortIdentifier 83
Table 28 – Function AbortReason 83
Table 29 – Function AbortDetail 84
Table 30 – DMPM state descriptions 84
Table 31 – DMPM state table – sender transactions 85
Table 32 – DMPM state table – receiver transactions 87
Table 33 – Primitives exchanged between data-link layer and DMPM 92
Table 34 – Function PickArep 94
Table 35 – Function FindAREP 95
Table 36 – Function LocateQubArep 95
Table 37 – Function SetIdentifier() 95

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-9: Application layer protocol specification – Type 9 elements

FOREWORD

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

NOTE 1 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 profile parts. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

International Standard IEC 61158-6-9 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This second edition cancels and replaces the first edition published in 2007. This edition constitutes a technical revision.

The main change with respect to the previous edition is listed below:

- Correction of Table 32.

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

FDIS	Report on voting
65C/607/FDIS	65C/621/RVD

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

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

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

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

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

NOTE 2 The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

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 6-9: Application layer protocol specification – Type 9 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 9 fieldbus. The term "time-critical" is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard defines in an abstract way the externally visible behavior provided by the Type 9 fieldbus Application Layer in terms of

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

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

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

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

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

1) define the wire-representation of the service primitives defined in IEC 61158-5-5, and

2) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the Type 9 IEC 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 61158-5-9.

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

1.3 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems. Conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

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

IEC 60559, *Binary floating-point arithmetic for microprocessor systems*

IEC 61158-3-1, *Industrial communication networks – Fieldbus specifications – Part 3-1: Data-link layer service definition – Type 1 elements*

IEC 61158-4-1, *Industrial communication networks – Fieldbus specifications – Part 4-1: Data-link layer protocol specification – Type 1 elements*

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

IEC 61158-5-9, *Industrial communication networks – Fieldbus specifications – Part 5-9: Application layer service definition – Type 9 elements*

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

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

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 Terms, definitions, symbols, abbreviations and conventions

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

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 Terms and definitions from ISO/IEC 7498-1

a) abstract syntax
b) application entity
c) application process
d) application protocol data unit
e) application service element
f) application entity invocation

g) application process invocation
h) application transaction
i) presentation context
j) real open system
k) transfer syntax

3.1.2 Terms and definitions from ISO/IEC 9545

a) application-association
b) application-context
c) application context name
d) application-entity-invocation
e) application-entity-type
f) application-process-invocation
g) application-process-type
h) application-service-element
i) application control service element

3.1.3 Terms and definitions from ISO/IEC 8824-1

a) object identifier
b) type
c) value
d) simple type
e) structured type
f) component type
g) tag
h) Boolean type
i) true
j) false
k) integer type
l) bitstring type
m) octetstring type
n) null type
o) sequence type
p) sequence of type
q) choice type
r) tagged type
s) any type
t) module
u) production

3.1.4 Terms and definitions from ISO/IEC 8825-1

a) encoding (of a data value)
b) data value
c) identifier octets (the singular form is used in this standard)
d) length octet(s) (both singular and plural forms are used in this standard)
e) contents octets

3.2 IEC/TR 61158-1 terms

The following IEC/TR 61158-1 terms apply.

3.2.1 application

function or data structure for which data is consumed or produced

3.2.2 application layer interoperability

capability of application entities to perform coordinated and cooperative operations using the services of the FAL

3.2.3 application object

object class that manages and provides the run time exchange of messages across the network and within the network device

NOTE: Multiple types of application object classes may be defined

3.2.4 application process

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

3.2.5 application process identifier

distinguishes multiple application processes used in a device

3.2.6 application process object

component of an application process that is identifiable and accessible through an FAL application relationship. Application process object definitions are composed of a set of values for the attributes of their class (see the definition for Application Process Object Class Definition). Application process object definitions may be accessed remotely using the services of the FAL Object Management ASE. FAL Object Management services can be used to load or update object definitions, to read object definitions, and to dynamically create and delete application objects and their corresponding definitions

3.2.7 application process object class

a class of application process objects defined in terms of the set of their network-accessible attributes and services

3.2.8 application relationship

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

3.2.9 application relationship application service element

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

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

3.2.11 attribute

description of an externally visible characteristic or feature of an object. 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

3.2.12 behaviour

indication of how the object responds to particular events. Its description includes the relationship between attribute values and services

3.2.13 class

a set of objects, all of which represent the same kind of system component. A class is a generalisation of the 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

3.2.14 class attributes

an attribute that is shared by all objects within the same class

3.2.15 class code

a unique identifier assigned to each object class

3.2.16 class specific service

a service defined by a particular object class to perform a required function which is not performed by a common service. A class specific object is unique to the object class which defines it

3.2.17 client

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

(b) an initiator of a message to which a server reacts, such as the role of an AR endpoint in which it issues confirmed service request APDUs to a single AR endpoint acting as a server

3.2.18 conveyance path

unidirectional flow of APDUs across an application relationship

3.2.19 cyclic

term used to describe events which repeat in a regular and repetitive manner

3.2.20 dedicated AR

AR used directly by the FAL User. On Dedicated ARs, only the FAL Header and the user data are transferred

3.2.21 device

a physical hardware connection to the link. A device may contain more than one node

3.2.22 device profile

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

3.2.23 dynamic AR

AR that requires the use of the AR establishment procedures to place it into an established state

3.2.24 endpoint

one of the communicating entities involved in a connection

3.2.25 error

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

3.2.26 error class

general grouping for error definitions. Error codes for specific errors are defined within an error class

3.2.27 error code

identification of a specific type of error within an error class

3.2.28 FAL subnet

networks composed of one or more data link segments. They are permitted to contain bridges, but not routers. FAL subnets are identified by a subset of the network address

3.2.29 logical device

specifies a certain FAL class that abstracts a software component or a firmware component as an autonomous self-contained facility of an automation device

3.2.30 management information

network-accessible information that supports managing the operation of the fieldbus system, including the application layer. Managing includes functions such as controlling, monitoring, and diagnosing

3.2.31 network

a series of nodes connected by some type of communication medium. The connection paths between any pair of nodes can include repeaters, routers and gateways

3.2.32 peer

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

3.2.33 pre-defined AR endpoint

AR endpoint that is defined locally within a device without use of the create service. Pre-defined ARs that are not pre-established are established before being used

3.2.34 pre-established AR endpoint

AR endpoint that is placed in an established state during configuration of the AEs that control its endpoints

3.2.35 publisher

role of an AR endpoint in which it transmits APDUs onto the fieldbus for consumption by one or more subscribers. The publisher may not be aware of the identity or the number of subscribers and it may publish its APDUs using a dedicated AR. Two types of publishers are defined by this standard, Pull Publishers and Push Publishers, each of which is defined separately

3.2.36 server

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

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

3.2.37 service

operation or function than an object and/or object class performs upon request from another object and/or object class. A set of common services is defined and provisions for the definition of object-specific services are provided. Object-specific services are those which are defined by a particular object class to perform a required function which is not performed by a common service

3.2.38 subscriber

role of an AREP in which it receives APDUs produced by a publisher. Two types of subscribers are defined by this standard, pull subscribers and push subscribers, each of which is defined separately

3.3 Abbreviations and symbols

AE	Application Entity
AL	Application Layer
ALME	Application Layer Management Entity
ALP	Application Layer Protocol
APO	Application Object
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Identifier
AR	Application Relationship
AREP	Application Relationship End Point
ASCII	American Standard Code for Information Interchange
ASE	Application service Element
Cnf	Confirmation
DL-	(as a prefix) data-link-
DLC	Data-link Connection
DLCEP	Data-link Connection End Point
DLL	Data-link layer
DLM	Data-link-management
DLSAP	Data-link service Access Point
DLSDU	DL-service-data-unit
FAL	Fieldbus Application Layer
ID	Identifier
IEC	International Electrotechnical Commission
Ind	Indication

LME	Layer Management Entity
OSI	Open Systems Interconnect
QoS	Quality of service
Req	Request
Rsp	Response
SAP	Service Access Point
SDU	Service Data Unit
SMIB	System Management Information Base
SMK	System Management Kernel
VFD	Virtual Field Device

3.4 Conventions

3.4.1 General concept

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

The class definitions define the attributes of the classes supported by each ASE. The attributes are accessible from instances of the class using the Management ASE services specified in IEC 61158-5 standard. The service specification defines the services that are provided by the ASE.

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

3.4.2 Conventions for class definitions

The data-link layer mapping definitions are described using templates. Each template consists of a list of attributes for the class. The general form of the template is defined in IEC 61158-5-5.

3.4.3 Abstract syntax conventions

When the "optionalParametersMap" parameter is used, a bit number which corresponds to each OPTIONAL or DEFAULT production is given as a comment.

3.5 Conventions used in state machines

The state machines are described in Table 1:

Table 1 – Conventions used for state machines

#	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 transaction. => 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 is taken.

The conventions used in the state machines are as follows:

:= Value of an item on the left is replaced by value of an item on the right. If an item on the right is a parameter, it comes from the primitive shown as an input event.

xxx A parameter name.

Example:
Identifier := reason
means value of a 'reason' parameter is assigned to a 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"

This construct allows the execution of a sequence of actions in a loop within one transition. The loop is executed for all values from start_value to end_value.

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

This construct allows the execution of alternative actions depending on some condition (which might be the value of some identifier or the outcome of a previous action) within one transition.

Example:
If (condition)
actions
else
actions
endif

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

4 Abstract syntax

4.1 FAL-AR PDU abstract syntax

4.1.1 Top level definition

The productions defined here shall be used with the rules for APDU encoding (see 5.1.2).

```
APDU ::= CHOICE {
            [PRIVATE 0] ConfirmedSend-RequestPDU,
            [PRIVATE 1] ConfirmedSend-ResponsePDU,
            [PRIVATE 2] UnconfirmedSend-PDU,
            [PRIVATE 3] UnconfirmedAcknowledgedSend-CommandPDU,
            [PRIVATE 4] Establish-RequestPDU,
            [PRIVATE 5] Establish-ResponsePDU,
            [PRIVATE 6] Establish-ErrorPDU,
            [PRIVATE 7] Abort-PDU,
            [PRIVATE 8] DataSendAcknowledge-PDU
}
```

4.1.2 Confirmed send service

```
ConfirmedSend-RequestPDU ::= SEQUENCE {
      [APPLICATION 0] AddressAREP,
      InvokeID,
      ConfirmedServiceRequest
}

ConfirmedSend-ResponsePDU ::= SEQUENCE {
      [APPLICATION 1] AddressAREP,
      InvokeID,
      pduBody     CHOICE {
         ConfirmedServiceResponse,
         ConfirmedServiceError
      }
}
```

4.1.3 Unconfirmed send service

```
UnconfirmedSend-PDU ::= SEQUENCE {
      [APPLICATION 2] AddressAREP,
      InvokeID,
      pduBody     CHOICE {
         UnconfirmedServiceRequest,
         UnconfirmedSendPD-PDU
      }
}
```

4.1.4 Unconfirmed acknowledge send service

```
UnconfirmedAcknowledgeSend-CommandPDU ::= SEQUENCE {
      [APPLICATION 3] AddressAREP,
      InvokeID,
      UnconfirmedServiceRequest
}
```

4.1.5 InvokeID

```
InvokeID ::= Unsigned8
```

4.1.6 Establish service

```
MaxOSCC ::= Unsigned8

MaxOSCS ::= Unsigned8

MaxUCSC ::= Unsigned8

MaxUCSS ::= Unsigned8

CIU ::= Unsigned32

Establish-RequestPDU ::= SEQUENCE {
      [APPLICATION 4] AddressAREP,
      ConType,
      MaxOSCC,
      MaxOSCS,
      MAXUCSC,
      MAXUCSS,
      CIU,
      InvokeID,
      initiateRequest                    [PRIVATE 0] IMPLICIT Initiate-RequestPDU
}
```

```
Establish-ResponsePDU ::= SEQUENCE {
        [APPLICATION 5] AddressAREP,
        InvokeID,
        initiateResponse                    [PRIVATE 0] IMPLICIT Initiate-ResponsePDU
}

Establish-ErrorPDU ::= SEQUENCE {
        [APPLICATION 6] AddressAREP,
        InvokeID,
        initiateError                       [PRIVATE 0] IMPLICIT Initiate-ErrorPDU
}
```

4.1.7 ConType

```
ConType ::= ENUMERATED {
    mmaz            (0)
}
```

4.1.8 Data send acknowledge service

```
DataSendAcknowledge-PDU ::= SEQUENCE {
        Protocol-Code,[APPLICATION 8],Address2ARP,          --- Protocol-Code in the higher nibble of the
octet!!!
        Block-Number,
        Block-Length,
        Protocol-Data,
}
```

4.1.9 Protocol-code

```
Protocol-Code ::= ENUMERATED {
    TCP/IP          (1)                         --- TCP/IP protocol
}
```

4.1.10 Block-number

```
Block-Number ::= Unsigned8                      --- 0  no blocking
                                                ---- 1..254 block number
                                                ---- 255 last block
```

4.1.11 Block-length

```
Block-Length ::= Unsigned
```

4.1.12 Protocol-data

```
Protocol-Data::= Any                            --- transparent protocol transfer
```

4.1.13 Address2 ARP

```
Address2ARP ::= SEQUENCE {
    Destination-Address,
    Source-Address,
    Destination-Node,
    Destination-Subnode,
    Source-Node,
    Source-Subnode
}
```

4.1.14 Destination-address

```
Destination-Address ::= OctetString                 --- 3 Octets
```

4.1.15 Source-address

```
Source-Address ::= OctetString                      --- 3 Octets
```

4.1.16 Destination-node

```
Destination-Node ::= Unsigned8
```

4.1.17 Source-node

Source-Node ::= Unsigned8

4.1.18 Destination-subnode

Destination-Subnode ::= Unsigned8

4.1.19 Source-subnode

Source-Subnode ::= Unsigned8

4.2 Abstract syntax of PDUBody

4.2.1 Abort service

```
Abort-PDU ::= SEQUENCE {
      [APPLICATION 7] AddressAREP,
      Identifier,
      ReasonCode,
      AdditionalDetail
}
```

4.2.2 ConfirmedServiceRequest

```
ConfirmedServiceRequest::= CHOICE {
   read-Request                  [0] IMPLICIT Read-RequestPDU,
   write-Request                 [3] IMPLICIT Write-RequestPDU,
   start-Request                 [6] IMPLICIT Start-RequestPDU,
   stop-Request                  [9] IMPLICIT Stop-RequestPDU,
   status-Request                [15] IMPLICIT Status-RequestPDU,
   identify-Request              [18] IMPLICIT Identify-RequestPDU,
   getAttributes1-Request        [21] IMPLICIT GetAttributes-RequestPDU,    -- short form
   getAttributes2-Request        [35] IMPLICIT GetAttributes-RequestPDU,    -- long form
   reset-Request                 [36] IMPLICIT Reset-RequestPDU,
   resume-Request                [39] IMPLICIT Resume-RequestPDU
}
```

4.2.3 ConfirmedServiceResponse

```
ConfirmedServiceResponse ::= CHOICE {
   read-Response                 [1] IMPLICIT Read-ResponsePDU,
   write-Response                [4] IMPLICIT Write-ResponsePDU,
   start-Response                [7] IMPLICIT Start-ResponsePDU,
   stop-Response                 [10] IMPLICIT Stop-ResponsePDU,
   status-Response               [16] IMPLICIT Status-ResponsePDU,
   identify-Response             [19] IMPLICIT Identify-ResponsePDU,
   getAttributes-Response        [22] IMPLICIT GetAttributes-ResponsePDU,
   reset-Response                [37] IMPLICIT Reset-ResponsePDU,
   resume-Response               [40] IMPLICIT Resume-ResponsePDU
}
```

4.2.4 ConfirmedServiceError

```
ConfirmedServiceError ::= CHOICE {
   read-Error                    [2] IMPLICIT ErrorType,
   write-Error                   [5] IMPLICIT ErrorType,
   start-Error                   [8] IMPLICIT ErrorFiType,
   stop-Error                    [11] IMPLICIT ErrorFiType,
   status-Error                  [17] IMPLICIT ErrorType,
   identify-Error                [20] IMPLICIT ErrorType,
   getAttributes-Error           [23] IMPLICIT ErrorType,
   reset-Error                   [38] IMPLICIT ErrorFiType,
   resume-Error                  [41] IMPLICIT ErrorFiType
}
```

4.2.5 Error type

Error type as specified in 4.1.4.6.

4.2.6 Error Fi type

```
ErrorFiType ::= SEQUENCE {
   errorClass                    [0] IMPLICIT ErrorClass,
   additionalCode                [1] IMPLICIT Integer16 OPTIONAL,
   fiState                       [3] IMPLICIT Integer8
}
```

4.2.7 Error class

Error Class as specified in 4.1.4.7.

4.2.8 Unconfirmed PDUs

```
UnconfirmedServiceRequest ::= CHOICE {
   informationReport-Request     [12] IMPLICIT InformationReport-RequestPDU,
   reject-Request                [34] IMPLICIT Reject-RequestPDU
}

UnconfirmedSendPD-PDU ::= BIT STRING
```

4.2.9 Management ASE

4.2.9.1 Get attributes service

```
GetAttributes-Request-PDU ::= SEQUENCE {
   listOfAttributes              [PRIVATE 0] IMPLICIT Mn_SelectedAttributes,
   accessSpecification  CHOICE {
      index                          [1] IMPLICIT Gn_NumericID,
      variableName               [2] IMPLICIT Gn_Name,
      fiName                     [5] IMPLICIT Gn_Name,
      startIndex                 [7] IMPLICIT Gn_NumericID
   }
}

GetAttributes-ResponsePDU ::= SEQUENCE {
   more                          [PRIVATE 0] IMPLICIT Gn_MoreFollows,
   listOfObjectDefinition        [PRIVATE 1] IMPLICIT SEQUENCE OF Gn_ObjectDefinition
}
```

4.2.10 Application process ASE

4.2.10.1 Get status service

```
Status-RequestPDU ::= NULL

Status-ResponsePDU ::= SEQUENCE {
   logicalStatus                 [PRIVATE 0] IMPLICIT ENUMERATED {
      ready-for-communication        (0),
      limited-services-permitted     (2),
   },
   physicalStatus                [PRIVATE 1] IMPLICIT ENUMERATED {
      operational                    (0),
      partially-operational          (1),
      inoperable                     (2),
      needs-commissioning            (3)
   },
   localDetail                   [PRIVATE 2] IMPLICIT BitString OPTIONAL
 }
```

4.2.10.2 Identify service

```
Identify-RequestPDU ::= NULL

Identify-ResponsePDU ::= SEQUENCE {
   vendorName                    [PRIVATE 0] IMPLICIT VisibleString,
   modelIdentifier               [PRIVATE 1] IMPLICIT VisibleString,
   vendorRevision                    [PRIVATE 2] IMPLICIT VisibleString
}
```

4.2.10.3 Initiate service

```
Initiate-RequestPDU ::= SEQUENCE {
   versionObjectDefinitionsCalling        [PRIVATE 0] IMPLICIT Integer16,
   apDescriptorCalling                    [PRIVATE 1] IMPLICIT OctetString,
   accessProtectionSupportedCalling       [PRIVATE 2] IMPLICIT Ap_AccessProtectionSupported,
   passwordAndAccessGroupsCalling         [PRIVATE 3] IMPLICIT Ap_AccessControl,
   configuredMaxPduSizeSending            [PRIVATE 4] IMPLICIT Unsigned8,
   configuredMaxPduSizeReceiving          [PRIVATE 5] IMPLICIT Unsigned8,
   listOfSupportedServicesCalling         [PRIVATE 6] IMPLICIT Mn_PduSupportedMap
}

Initiate-ResponsePDU ::= SEQUENCE {
   versionObjectDefinitionsCalled         [PRIVATE 0] IMPLICIT Integer16,
   apDescriptorCalled                     [PRIVATE 1] IMPLICIT OctetString,
   accessPrivilegeSupportedCalled         [PRIVATE 2] IMPLICIT Ap_AccessProtectionSupported,
   passwordAndAccessGroupsCalled          [PRIVATE 3] IMPLICIT Ap_AccessControl
}

Initiate-ErrorPDU ::= SEQUENCE {
   errorCode                              [PRIVATE 0] IMPLICIT  ENUMERATED {
      other                                  (0),
      max-fal-pdu-size-insufficient          (1),
      service-not-supported                  (2),
      version-obj-def-incompatible           (3),
      user-initiate-denied                   (4),
      password-error                         (5),
      profile-number-incompatible            (6)
      },
   maxPduLengthSendingCalled              [PRIVATE 1] IMPLICIT Unsigned8,
   maxPduLengthReceivingCalled            [PRIVATE 2] IMPLICIT Unsigned8,
   listOfSupportedServicesCalled          [PRIVATE 3] IMPLICIT Mn_PduSupportedMap
 }
```

4.2.10.4 Reject service

```
Reject-RequestPDU  ::=  SEQUENCE  {
   original-invokeID                      [PRIVATE 0] IMPLICIT Integer8,
   reject-code                            [PRIVATE 1] IMPLICIT ENUMERATED {
      pdu-size                               (5)
   }
}
```

4.2.11 Function invocation ASE

4.2.11.1 Reset service

```
Reset-RequestPDU ::= SEQUENCE {
   keyAttribute                           Gn_KeyAttribute
}

Reset-ResponsePDU ::= NULL
```

4.2.11.2 Resume service

```
Resume-RequestPDU ::= SEQUENCE {
   keyAttribute                           Gn_KeyAttribute
}

Resume-ResponsePDU ::= NULL
```

4.2.11.3 Start service

```
Start-RequestPDU ::= SEQUENCE {
   keyAttribute                           Gn_KeyAttribute
}

Start-ResponsePDU ::= NULL
```

4.2.11.4 Stop service

```
Stop-RequestPDU ::= SEQUENCE {
   keyAttribute                           Gn_KeyAttribute
}

Stop-ResponsePDU ::= NULL
```

4.2.12 Variable ASE

4.2.12.1 Information report service

```
InformationReport-RequestPDU ::= SEQUENCE {
   index                              Gn_NumericID,
   subIndex                           [PRIVATE 0] IMPLICIT Gn_SubIndex OPTIONAL,
   value                              [PRIVATE 1] IMPLICIT ANY
}
```

4.2.12.2 Read service

```
Read-RequestPDU ::= SEQUENCE {
   index                              Gn_NumericID,
   subIndex                           [PRIVATE 0] IMPLICIT Gn_SubIndex OPTIONAL
}

Read-ResponsePDU ::= SEQUENCE {
   value                                 [PRIVATE 0] IMPLICIT ANY
}
```

4.2.12.3 Write service

```
Write-RequestPDU ::= SEQUENCE {
   index                              Gn_NumericID,
   subIndex                           Gn_SubIndex OPTIONAL,
   value                              [PRIVATE 0] IMPLICIT ANY
}

Write-ResponsePDU ::= NULL
```

4.3 Type definitions for ASEs

4.3.1 AP ASE types

4.3.1.1 Ap_AccessProtectionSupported

```
Ap_AccessProtectionSupported ::= Boolean        -- True means Access Protection is supported.
                                                -- False means Access Protection is not supported.
```

4.3.1.2 Ap_AccessControl

```
Ap_AccessControl ::= BitString {          -- The Password (Unsigned8) is encoded as a bit string.
   password_Bit1                          (8),
   password_Bit2                          (7),
   password_Bit3                          (6),
   password_Bit4                          (5),
   password_Bit5                          (4),
   password_Bit6                          (3),
   password_Bit7                          (2),
   password_Bit8                          (1),
   access_Groups-1                        (16),
   access_Groups-2                        (15),
   access_Groups-3                        (14),
   access_Groups-4                        (13),
   access_Groups-5                        (12),
   access_Groups-6                        (11),
   access_Groups-7                        (10),
   access_Groups-8                        (9)
}
```

4.3.2 AR ASE types

4.3.2.1 Reason code and additional detail

4.3.2.1.1 Reason code

```
ReasonCode ::= Unsigned8
```

4.3.2.1.2 Additional detail

```
AdditionalDetailI ::= OctetString
```

4.3.2.2 AREP

```
AddressAREP::= Unsigned8                  -- Least significant octet of DLCEP address
```

4.3.2.3 Abort type identifier

```
Identifier ::= ENUMERATED {
   fal-user                    (0),
   fal-APO-ASE                 (1),
   fal-AR-ASE                  (2),
   dll                         (3)
}
```

4.3.3 Function Invocation ASE types

4.3.3.1 Fi_AccessPrivilege

```
Fi_AccessPrivilege ::= BitString {
   rightToStartPassword        (24),
   rightToStopPassword         (23),
   rightToDeletePassword       (22),
   rightToStartAccessGroup     (20),
   rightToStopAccessGroup      (19),
   rightToDeleteAccessGroup    (18),
   rightToStartAllPartner      (32),
   rightToStopAllPartner       (31),
   rightToDeleteAllPartner     (30),
   password_Bit1               (8),    -- The Password (Unsigned8) is encoded as a bit string.
   password_Bit2               (7),
   password_Bit3               (6),
   password_Bit4               (5),
   password_Bit5               (4),
   password_Bit6               (3),
   password_Bit7               (2),
   password_Bit8               (1),
   access_Groups-1             (16),
   access_Groups-2             (15),
   access_Groups-3             (14),
   access_Groups-4             (13),
   access_Groups-5             (12),
   access_Groups-6             (11),
   access_Groups-7             (10),
   access_Groups-8             (9)
}
```

4.3.3.2 Fi_State

```
Fi_State ::= Unsigned8 {
   unrunnable                  (1),
   idle                        (2),
   running                     (3),
   stopped                     (4),
   starting                    (5),
   stopping                    (6),
   resuming                    (7),
   resetting                   (8)
}
```

4.3.4 Management ASE types

4.3.4.1 Mn_PduSupportedMap

```
Mn_PduSupportedMap ::= BIT STRING {
   getAttributes-RequestPDU    (1),    -- Requester
   start-RequestPDU            (8),
   stop-RequestPDU             (9),
   resume-RequestPDU           (9),
   reset-RequestPDU            (9),
   read-RequestPDU             (11),
   write-RequestPDU            (12),
   informationReport-RequestPDU (17),
   getAttributes-ResponsePDU   (25),   -- Responder
   start-ResponsePDU           (33),
   stop-ResponsePDU            (33),
   resume-ResponsePDU          (33),
   reset-ResponsePDU           (33),
   read-ResponsePDU            (35),
   write-ResponsePDU           (36),
   informationReport-ResponsePDU (41)
}
```

4.3.5 Variable ASE types

4.3.5.1 Vr_AccessPrivilege

```
Vr_AccessPrivilege ::= BitString {
   rightToReadPassword              (24),
   rightToWritePassword             (23),
   rightToDeletePassword            (22),
   rightToReadAccessGroup           (20),
   rightToWriteAccessGroup          (19),
   rightToDeleteAccessGroup         (18),
   rightToReadAllPartners           (32),
   rightToWriteAllPartners          (31),
   rightToDeleteAllPartners         (30),
   password_Bit1                    (8),     -- The Password (Unsigned8) is encoded as a bit string.
   password_Bit2                    (7),
   password_Bit3                    (6),
   password_Bit4                    (5),
   password_Bit5                    (4),
   password_Bit6                    (3),
   password_Bit7                    (2),
   password_Bit8                    (1),
   access_Groups-1                  (16),
   access_Groups-2                  (15),
   access_Groups-3                  (14),
   access_Groups-4                  (13),
   access_Groups-5                  (12),
   access_Groups-6                  (11),
   access_Groups-7                  (10),
   access_Groups-8                  (9)
}
```

4.3.6 General types

4.3.6.1 Gn_Deletable

```
Gn_Deletable ::= Boolean            -- True means deletable.
                                    -- False means not deletable.
```

4.3.6.2 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
}
```

4.3.6.3 Gn_Length

```
Gn_Length ::= Unsigned8
```

4.3.6.4 Gn_MoreFollows

```
Gn_MoreFollows ::= Boolean
```

4.3.6.5 Gn_NumericID

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

4.3.6.6 Gn_Name

```
Gn_Name  ::=  OctetString
```

4.3.6.7 Gn_ObjectClass

```
Gn_ObjectClass ::= ENUMERATED {
   functionInvocation                (3),
   fixedLengthStringDataType         (5),
   structuredDataType                (6),
   fixedLengthStringVariable         (7),
   arrayVariable                     (8),
   dataStructureVariable             (9),
}
```

4.3.6.8 Gn_ObjectDefinition

```
Gn_ObjectDefinition ::= OctetString      -- The semantics of this parameter are application specific.
```

4.3.6.9 Gn_Reusable

```
Gn_Reusable ::= Boolean                  -- True means reusable.
                                         -- False means not reusable.
```

4.3.6.10 Gn_SubIndex

```
Gn_SubIndex ::= Unsigned8
```

4.3.6.11 Gn_TypeDescription

```
Gn_TypeDescription  ::=  CHOICE {
   boolean                              [1]  Gn_Length,
   integer8                             [2]  Gn_Length,
   integer16                         [3]  Gn_Length,
   integer32                         [4]  Gn_Length,
   unsigned8                         [5]  Gn_Length,
   unsigned16                        [6]  Gn_Length,
   unsigned32                        [7]  Gn_Length,
   float                             [8]  Gn_Length,
   visiblestring                     [9]  Gn_Length,
   octetstring                       [10] Gn_Length,
   binaryDate                        [11] Gn_Length,
   timeOfDay                         [12] Gn_Length,
   timeDifference                    [13] Gn_Length,
   bitstring                            [14] Gn_Length
}
```

4.3.7 Object definitions

4.3.7.1 Top Level Definition

```
Object-Definition ::= CHOICE {
            [PRIVATE 0] ListHeader,
            [PRIVATE 1] DataTypeList,
            [PRIVATE 2] StaticList,
            [PRIVATE 3] FunctionInvocationDefinition
}
```

4.3.7.2 ListHeader

```
ListHeader ::= SEQUENCE {
   numericId                                  [PRIVATE 0] IMPLICIT Unsigned16,
   romRamFlag                                 [PRIVATE 1] IMPLICIT Boolean,
   maxNameLength                              [PRIVATE 2] IMPLICIT Unsigned8,
   accessProtectionSupported                  [PRIVATE 3] IMPLICIT Boolean,
   versionOfObjectDefinition                  [PRIVATE 4] IMPLICIT Integer16,
   localReferenceOfListHeader                 [PRIVATE 5] IMPLICIT Unsigned32 OPTIONAL,
   numberOfEntriesInDataTypeList              [PRIVATE 6] IMPLICIT Unsigned16,
   localReferenceOfDataTypeList               [PRIVATE 7] IMPLICIT Unsigned32 OPTIONAL,
   firstNumericIdOfStaticList                 [PRIVATE 8] IMPLICIT Unsigned16,
   numberOfEntriesInStaticList                [PRIVATE 9] IMPLICIT Unsigned16,
   localReferenceOfStaticList                 [PRIVATE 10] IMPLICIT Unsigned32 OPTIONAL,
   firstNumericIdOfVariableListDefinition     [PRIVATE 11] IMPLICIT Unsigned16,
   numberOfEntriesInVariableListDefinition        [PRIVATE 12] IMPLICIT Unsigned16,
   localReferenceOfVariableListDefinition     [PRIVATE 13] IMPLICIT Unsigned32 OPTIONAL,
   firstNumericIdOfFunctionInvocationDefinition [PRIVATE 14] IMPLICIT Unsigned16,
   numberOfEntriesInFunctionInvocationDefinition [PRIVATE 15] IMPLICIT Unsigned16,
   localReferenceOfFunctionInvocationDefinition [PRIVATE 16] IMPLICIT Unsigned32 OPTIONAL
}
```

4.3.7.3 DataTypeList

```
DataTypeList ::= CHOICE {
        [PRIVATE 0] DataTypeDefinition,
            [PRIVATE 1] StructuredDataTypeDefinition
}

DataTypeDefinition ::= SEQUENCE {
   numericId                                Gn_NumericID,
   objectClass                              Gn_ObjectClass,
   dataTypeNameLength                       Gn_Length,
   dataTypeName                             [PRIVATE 0] IMPLICIT VisibleString OPTIONAL
}

StructuredDataTypeDefinition ::= SEQUENCE {
   numericId                                Gn_NumericID,
   objectClass                              Gn_ObjectClass,
   numberOfElements                    [PRIVATE 0] IMPLICIT Integer8,
   recordList      SEQUENCE OF SEQUENCE{
      numericIdOfDataTypeDefinition         Gn_NumericID,
      dataLength                            Gn_Length
   }
}
```

4.3.7.4 StaticList

```
StaticList ::= CHOICE {
        [PRIVATE 0] VariableDefinition,
        [PRIVATE 1] ArrayDefinition,
        [PRIVATE 2] StructureDefinition
}

VariableDefinition ::= SEQUENCE {
   numericId                                Gn_NumericID,
   objectClass                              Gn_ObjectClass,
   numericIdOfDataTypeDefinition            Gn_NumericID,
   dataLength                               Gn_Length,
   accessPrivilege                          Vr_AccessPrivilege OPTIONAL,
   localReferenceOfVariable            [PRIVATE 0] IMPLICIT Unsigned32 OPTIONAL,
   variableName                        [PRIVATE 1] IMPLICIT VisibleString OPTIONAL,
   extension                           [PRIVATE 2] IMPLICIT OctetString OPTIONAL
}

ArrayDefinition ::= SEQUENCE {
   numericId                                Gn_NumericID,
   objectClass                              Gn_ObjectClass,
   numericIdOfDataTypeDefinition            Gn_NumericID,
   dataLength                               Gn_Length,
   numberOfElements                    [PRIVATE 0] IMPLICIT Unsigned8,
   accessPrivilege                          Vr_AccessPrivilege OPTIONAL,
   localReferenceOfArray               [PRIVATE 1] IMPLICIT Unsigned32 OPTIONAL,
   arrayName                           [PRIVATE 2] IMPLICIT VisibleString OPTIONAL,
   extension                           [PRIVATE 3] IMPLICIT OctetString OPTIONAL
}

StructureDefinition ::= SEQUENCE {
   numericId                                Gn_NumericID,
   objectClass                              Gn_ObjectClass,
   numericIdOfDataTypeDefinition            Gn_NumericID,
   accessPrivilege                          Vr_AccessPrivilege OPTIONAL,
   structureName                       [PRIVATE 0] IMPLICIT VisibleString OPTIONAL,
   extension                           [PRIVATE 1] IMPLICIT OctetString OPTIONAL,
   localReferenceOfElement             [PRIVATE 2] IMPLICIT SEQUENCE OF Unsigned32 OPTIONAL
}
```

4.3.7.5 FunctionInvocationDefinition

```
FunctionInvocationDefinition ::= SEQUENCE {
   numericId                          Gn_NumericID,
   objectClass                        Gn_ObjectClass,
   numberOfRelatedObjects             [PRIVATE 0] IMPLICIT Unsigned8,
   accessPrivilege                    Fi_AccessPrivilege OPTIONAL,
   deletable                          Gn_Deletable,
   reusable                           Gn_Reusable,
   functionInvocationState            FI_State,
   numericIdOfLoadRegion              SEQUENCE OF Gn_NumericID,
   functionInvocationName             [PRIVATE 1] IMPLICIT VisibleString OPTIONAL,
   extension                          [PRIVATE 2] IMPLICIT OctetString OPTIONAL
}
```

4.4 Abstract syntax of data types

4.4.1 Referenced data types

4.4.2 Notation for the Boolean type

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

4.4.3 Notation for the Integer types

```
Integer ::= INTEGER                    -- any integer
Integer8 ::= INTEGER  (-128..+127)     -- range  -2^7 <= i <= 2^7-1
Integer16 ::= INTEGER  (-32768..+32767) -- range  -2^15 <= i <= 215-1
Integer32 ::= INTEGER                  -- range  -2^31 <= i <= 2^31-1
```

4.4.4 Notation for the Unsigned types

```
Unsigned ::= INTEGER                   -- any non-negative integer
Unsigned8 ::= INTEGER  (0..255)        -- range 0 <= i <= 2^8-1
Unsigned16 ::= INTEGER  (0..65535)     -- range 0 <= i <= 2^16-1
Unsigned32 ::= INTEGER                 -- range 0 <= i <= 2^32-1
```

4.4.5 Notation for the Floating Point type

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

4.4.6 Notation for the BitString type

```
BitString ::= BIT STRING               -- For generic use
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
```

4.4.7 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
```

4.4.8 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
```

4.4.9 Notation for BinaryDate type

```
BinaryDate ::= OctetString7
```

4.4.10 Notation for TimeOfDay type

```
TimeOfDay ::= OctetString6
```

4.4.11 Notation for TimeDifference type

TimeDifference ::= OctetString6

4.4.12 Notation for TimeValue type

TimeValue ::= OctetString8

4.4.13 Nortation for DL—Time-offset type

DL-Time-offsetType ::= OctetString

5 Transfer syntax

5.1.1 General

Additional information is to be added to the user data to allow an unique association of the data at the communication partner. The coding rules for the additional information are optimized to produce messages as short as possible in accordance with fieldbus requirements. The frequency of occurrence of special messages is taken into account.

The conditions in the fieldbus area are the following:

- short messages
- low number of different messages
- some messages such as Read and Write occur especially often.

5.1.2 Coding rules

The structuring of a FMS PDU is done either by inserting explicitly Identification Information or by implicit agreements.

Structure of a PDU is shown in Figure 1.

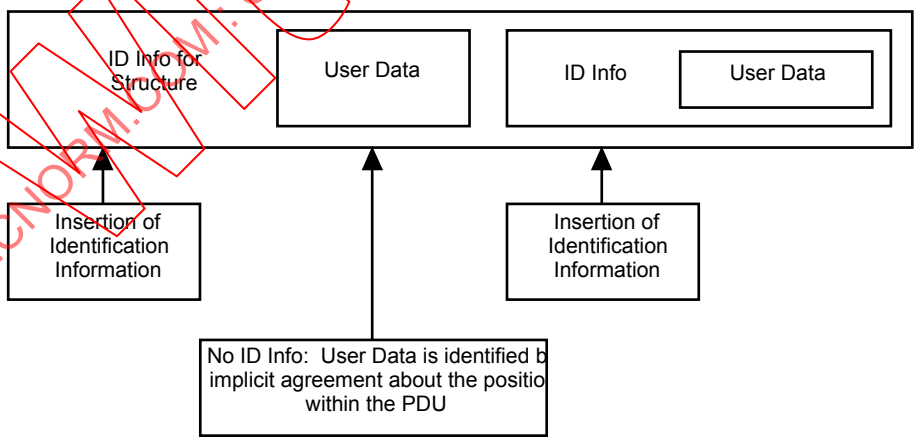


Figure 1 – Insertion of identification information in the FMS PDU

The Identification Information consists of P/C flag, tag and length. The structures and the user data of the PDU may be identified using this Identification Information.

The semantics of the user data are either known from the context or are universally known (context specific tags or universal tags). In the syntax description the context specific tags are

enclosed in rectangular brackets. If the semantics of a parameter are implicitly known from the position in the PDU, no tag is used.

The following restrictions on the usage of implicitly known components (universal tags) shall be made:

- the length of the user data shall be fixed,
- the component may not be OPTIONAL,
- the component may not be inside a CHOICE construct.

5.1.3 Structure of the identification information

5.1.3.1 General

The Identification Information (ID Info) consists of P/C flag, tag and length, as shown in Figure 2.

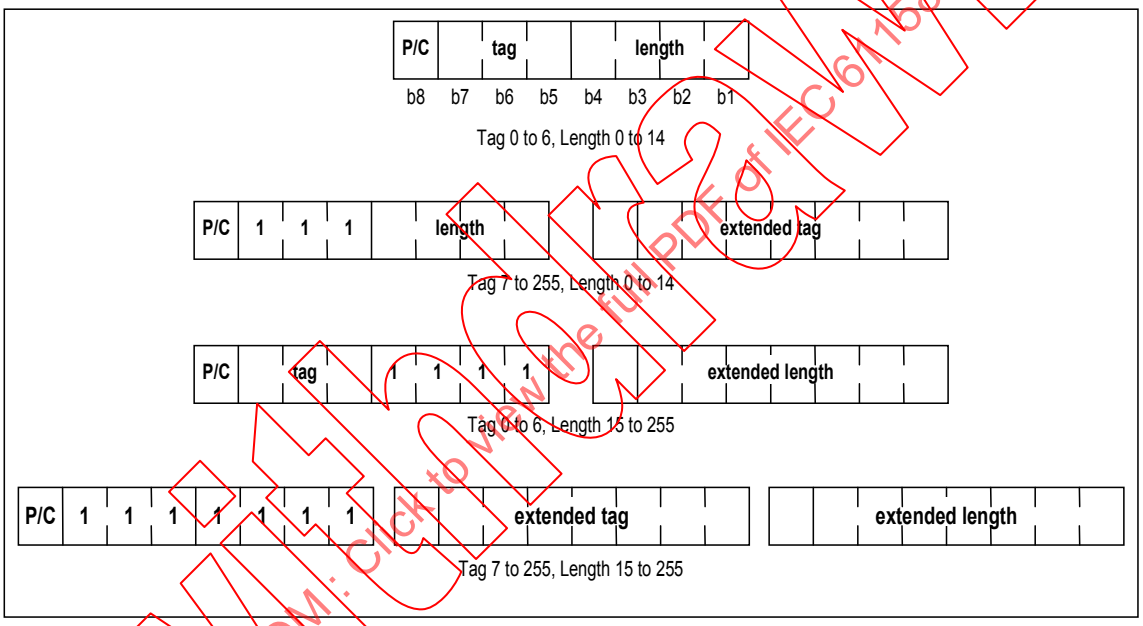


Figure 2 – Identification

The P/C flag indicates if it is a simple component (primitive) or if it is a structured component (constructed, SEQUENCE, SEQUENCE OF).

P/C Flag =0 <=> simple component

P/C Flag =1 <=> structured component

The tag identifies the semantics of the component.

The length is the length of the component in octets if this component is a simple one or the number of contained components if this component is structured.

The whole ID Info is coded in one octet, if possible. The octet is divided into 3 parts of different lengths.

- P/C flag 1 bit
- tag 3 bits
- length 4 bits

If the space in the fields for the length or for the tag is not sufficient, an extension is used. For this all bits of the concerned field are set to one. The information is then encoded in the following octet. The range of the tag is 0 to 6 and the range of the length is 0 to 14 when no extension is used. These ranges are preferred for the most frequently occurring messages because they produce very short PDUs.

If an extension has to be used for both the tag and the length, the tag is encoded in the first subsequent octet and the length is encoded in the second subsequent octet.

5.1.3.2 Data types

5.1.3.2.1 General

User data is always a simple (primitive) component. It is encoded as shown in Figure 3.

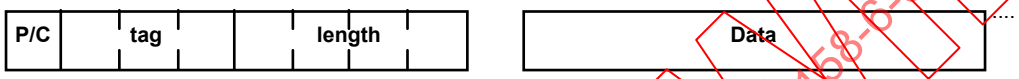


Figure 3 – Coding with identification

If the semantics of the user data are known implicitly from the position in the PDU and the length is fixed and implicitly known, then no identification information is added as shown in Figure 4.

Figure 4 – Coding without identification

5.1.3.2.2 Coding for Boolean type

Representation of the value true or false in one octet as shown in Figure 5 and Figure 6.

Notation: Range of values:	Boolean
Coding:	true or false false is represented by the value 0, true is represented by the value FF

	bits	8	7	6	5	4	3	2	1			
	octet 1	1	1	1	1	1	1	1	1			

Figure 5 – Representation of the value true

bits	8	7	6	5	4	3	2	1
octet 1	0	0	0	0	0	0	0	0

Figure 6 – Representation of the value false

5.1.3.2.3 Coding for Integer types

Integer values are signed quantities as shown in Figure 7.

Notation:	Integer8, Integer16, Integer32		
Range of Values:	Data type	range of values	length
	Integer8	-128 ≤ i ≤ 127	1 octet
	Integer16	-32768 ≤ i ≤ 32767	2 octets
	Integer32	$-2^{31} \le i \le 2^{31}-1$	4 octets
Coding:	In two's complement representation the MSB (Most Significant Bit) is the bit after the sign bit (SN) in the first octet. SN = 0: positive numbers and z SN = 1: negative numbers		

bits	8	7	6	5	4	3	2	1
octets 1	SN	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Figure 7 – Coding of data of data type Integer16

5.1.3.2.4 Coding for Unsigned types

Unsigned Values are encoded as shown in Figure 8.

Notation:	Unsigned8, Unsigned16, Unsigned32		
Range of Values:	**Data type**	**range of values**	**length**
	Unsigned8	$0 \le i \le 255$	1 octet
	Unsigned16	$0 \le i \le 65535$	2 octets
	Unsigned32	$0 \le i \le 4294967295$	4 octets

Coding	Binary							
bits	8	7	6	5	4	3	2	1
octets								
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

Figure 8 – Coding of data of data type Unsigned16

5.1.3.2.5 Coding for Floating Point type

Floating Point values are encoded as shown in Figure 9.

Notation:	Floating-Point (4 octet)
Range of Values:	see IEEE Std 754 Short Real Number (32 bits)
Coding:	see IEEE Std 754 Short Real Number (32 bits)

								LSB
bits	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
octets	**Exponent (E)**							
1	SN	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}
	(E)	**Fraction (F)**						
2	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
	Fraction (F)							
3	2^{-8}	2^{-9}	2-10	2-11	2-12	2-13	2-14	2-15
	Fraction (F)							
4	2-16	2-17	2-18	2-19	2-20	2-21	2-22	2^{-23}

SN: sign 0 = positive, 1 = negative

Figure 9 – Coding of data of data type Floating Point

5.1.3.2.6 Coding for Visible String type

Visible String values are encoded as shown in Figure 10.

Notation:	Visible-String
Range of Values:	see ISO 646 and ISO/IEC 2375: Defining
registration	
	number 2 + SPACE
Coding:	see ISO 646

bits	8	7	6	5	4	3	2	1
octets								
1	first character							
2	second character							
.	etc.							
.								
n								

Figure 10 – Coding of data of data type Visible String

5.1.3.2.7 Coding for Octet String type

Octet String values are encoded as shown in Figure 11.

Notation: Octet String
Coding: Binary

bits	8	7	6	5	4	3	2	1
octets								
1								
n								

Figure 11 – Coding of data of data type Octet String

5.1.3.2.8 Coding for Date type

The data type Date consists of a calendar date and a time as shown in Table 2 and Figure 12.

Notation: Date/Time
Range of Values: ms to 99 years
Coding: in 7 octets

Table 2 – Coding for Date type

Parameter	Range of values	Meaning of the parameters
ms	0…59 999	milliseconds
min	0…59	minutes
SU	0,1	0: standard time, 1: summer time
RSV	—	reserve
h	0…23	hours
d. of w.	1…7	day of week: 1 = Monday, 7 = Sunday
d. of m.	1…31	day of month
months	1…12	months
years	0…99	years (without the century)

bits	8	7	6	5	4	3	2	1	
octets									
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0…59 999 ms
3	RSV	RSV	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0…59 min
4	SU	RSV	RSV	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0…23 h
5	day of week			day of month					1…7 d. of w.
	2^{2}	2^{1}	2^{0}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	1…31 d. of m.
6	RSV	RSV	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	1…12 months
7	RSV	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	0 … 99 years
	msb								

Figure 12 – Coding of data of type Date

5.1.3.2.9 Coding for Time-of-day type

The data type Time-of-day consists of a time and an optional date.

The time is stated in milliseconds since midnight. At midnight the counting starts with the value zero.

The date is stated in days relatively to the first of January 1984. On the first of January 1984 the date starts with the value zero.

Notation: Time-of-day

Range of Values: $0 \le i \le (2^{28} -1)$ ms
$0 \le i \le (2^{16} -1)$ days

Coding:

The time is represented as a 32 bit binary value. The first four (MSB) bits shall have the value zero.

The (optional) date is encoded as a 16 bit (2 octets) binary value.

The Time-of-day is a string of 4 or 6 octets as shown in Figure 13.

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

Figure 13 – Coding of data of data type Time-of-day

5.1.3.2.10 Coding for Time-difference type

The data type Time-difference consists of a time in milliseconds and an optional day count. The structure is equivalent to the structure of the Time-of-day but it states in this case a Time Difference.

Notation Time-difference

Range of Values: $0 \leq i \leq (2^{32} - 1)$ ms
$0 \leq i \leq (2^{16} - 1)$ days

Coding:

The time is represented as a 32 bit binary value. The first four (MSB) bits shall have the value zero. The (optional) date is encoded as a 16 bit (2 octets) binary value. The Time Difference is a string of 4 or 6 octets as shown in Figure 14.

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

Figure 14 – Coding of data of data type Time-difference

5.1.3.2.11 Coding for Bit String type

Figure 15 shows the numbering scheme of the bits of the data type Bit String.

Only multiples of eight are legal values of the length (in bits) of the Bit String.

Notation: Bit String
Coding: Binary

bits	8	7	6	5	4	3	2	1
octets								
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
.	etc.							
n								

Figure 15 – Coding of data of data type Bit String

5.1.3.2.12 Coding for Time-value type

This data type is used to represent date and time in the required precision for application clock synchronization. It is a 64-bit unsigned fixed-point number, which is the time in 1/32s of a millisecond. When SN bit is 1, the data is negative and in two's complement representation, while the data is positive when SN bit is 0. See Figure 16.

bits	8	7	6	5	4	3	2	1	
octets 1	SN	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}	
2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}	
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}	
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	signed integer
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	of 8 octets length of 1/32ms unit
	msb								

Figure 16 – Coding of data of data type Time-value

5.1.3.3 User data definitions

5.1.3.3.1 General

User data is always a simple (primitive) component. It is encoded as shown in Figure 16. The P/C, tag, and length are encoded as shown in Figure 17.

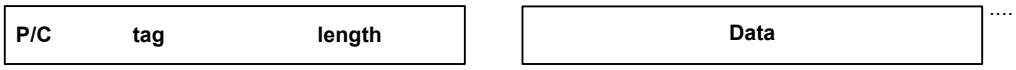


Figure 17 – Coding of data of user data definitions with identifier

If the semantics of the user data are known implicitly from the position in the PDU and the length is fixed and implicitly known, then no Identification Information is added as shown in Figure 18.

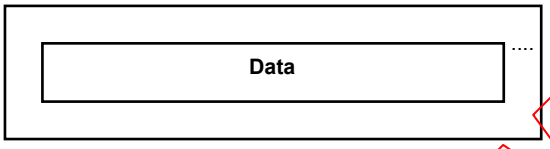


Figure 18 – Coding of data of user data definitions without identifier

Two special definitions are used to describe user data in the PDUBody definitions, Null and Packed. They are defined below.

5.1.3.3.2 Coding for Null

Null has the length zero. There are no subsequent (empty) octets.

5.1.3.3.3 Coding for Packed

Packed contains one or more data elements of the Data types which are chained together without any gap. The composition is known by the user.

5.1.3.4 Coding of structure information

5.1.3.4.1 General

User data may be combined to structured (constructed) components.

The communication partner shall be able to identify these structures and the components of these structures. The P/C flag of the ID Info is 1.

5.1.3.4.2 SEQUENCE

The SEQUENCE structure is comparable with a record. It represents a collection of user data of the same or of different Data types. Before the SEQUENCE structure there is an ID Info, which conveys the length not in octets but in number of components. The number of components is less than the total length in octets. In most cases an extension is not necessary due to this length encoding. Components should be encoded in the order of ANS.1. See Figure 19. Neither duplication nor deletion of an item is allowed.

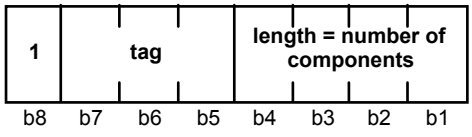


Figure 19 – Coding of ID info for a SEQUENCE

A structure may contain user data or further structures as components. Single components may be OPTIONAL, i.e. they may be omitted. In this case the ID Info is omitted too. A SEQUENCE shall be counted as a single component even if it contains several components.

Example:

The hexadecimal notation is used for the following example of encoding. The upper case letter X is used as a fill-in for unknown values, such as the length of the single components or the tag of the structure. A lower case letter x represents user data.

Syntax Description	Code	comment
Person [1] IMPLICIT SEQUENCE	{94	4 components
[0] Surname,	0X xx ...	tag 0
[1] First name,	1X xx ...	tag 1
[2] City,	2X xx ...	tag 2
[3] Street	3X xx ...	tag 3
}		

5.1.3.4.3 SEQUENCE OF

The SEQUENCE OF structure represents a succession of components. It is comparable with an array.

The structure may contain one or more components. The components may be user data or structures.

The coding is as for the structure SEQUENCE. For the statement of the number of components the number of repetitions shall be taken into account. The tags of the Syntax Description shall be used for the components of SEQUENCE OF. The same tags may be coded several times in succession.

Example:

```
employeedata [2] IMPLICIT SEQUENCE OF {
   [0] Person
}
```

5.1.3.4.4 CHOICE

A CHOICE represents a selection from a set of predefined possibilities. The components of a CHOICE construct shall have different tags to allow proper identification. Instead of the CHOICE construct the actually selected component is encoded. Only one component should be encoded for a CHOICE. Its tag has a value specified in ASN.1.

Example:

```
Data ::= CHOICE {
   [0] Employeedata,
   [1] Clientdata,
   [2] Supplierdata
}
```

6 Structure of FAL protocol state machines

Interface to FAL services and protocol machines are specified in this subclause.

NOTE The state machines specified in this subclause and ARPMs defined in the following sections only define the valid events for each. It is a local matter to handle these invalid events.

The behavior of the FAL is described by three integrated protocol machines. Specific sets of these protocol machines are defined for different AREP types. The three protocol machines are: FAL service Protocol Machine (FSPM), the Application Relationship Protocol Machine (ARPM), and the data-link layer Mapping Protocol Machine (DMPM). The relationship among these protocol machines as well as primitives exchanged among them are depicted in Figure 20.

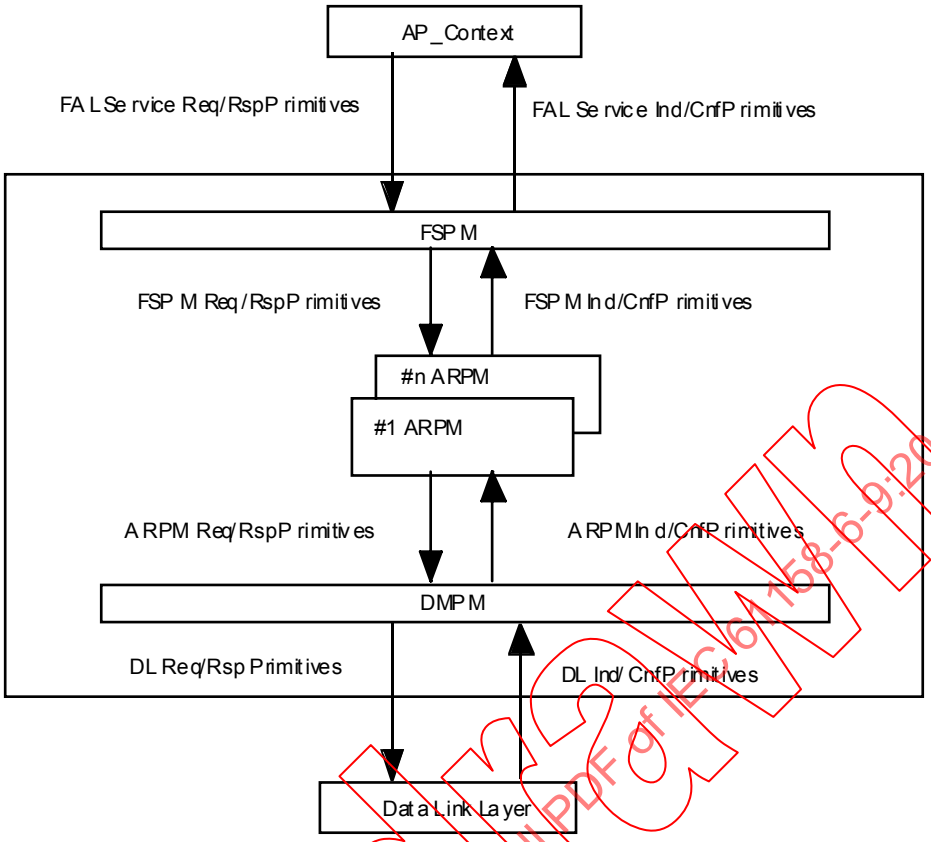


Figure 20 – Relationships among protocol machines and adjacent layers

The FSPM describes the service interface between the AP-Context and a particular AREP. The FSPM is common to all the AREP classes and does not have any state changes. 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 describes the establishment and release of an AR and exchange of FAL-PDUs with a remote ARPM(s). The ARPM is responsible for the following activities:

e) 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;

f) to accept FAL internal primitives from the DMPM and send them to the FSPM as a form of FAL internal primitives;

g) 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:

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

i) to receive DLL indication or confirmation primitives from the DLL and send them to the ARPM in a form of FAL internal primitives.

7 AP-Context state machines

7.1 VCR PM structure

This fieldubs defines a single state machine, the the VCR Connection protocol machine. Its relationship with its user and with the underlying FSPM, as well as primitives exchanged among them, are depicted in Figure 21.

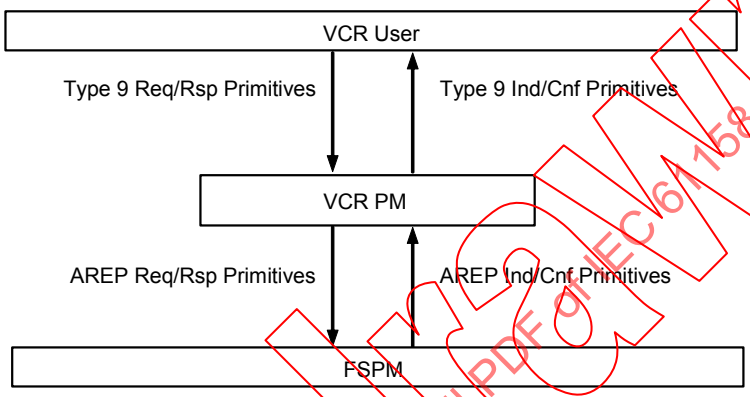


Figure 21 – Relationships among protocol machines and adjacent layers

The VCR Protocol Machine (VCR PM) is an VCR PM state machine that describes the operation of VCRs that are dynamically established by exchanging Initiate service APDUs. The VCR PM is responsible for the following activities:

1. to accept service request and response primitives from the VCR service user;
2. to accept service indication and confirmation primitives from the ARPMs;
3. to deliver service indication and confirmation primitives to the VCR user.
4. to issue service request and response primitives to the ARPM based on the rules of its state machine.

7.2 VCR PM state machine

7.2.1 General

The VCR PM state machine, illustrated in Figure 22, describes the VCR endpoint behavior. For connectionless VCRs, the state machine begins in the OPEN state.

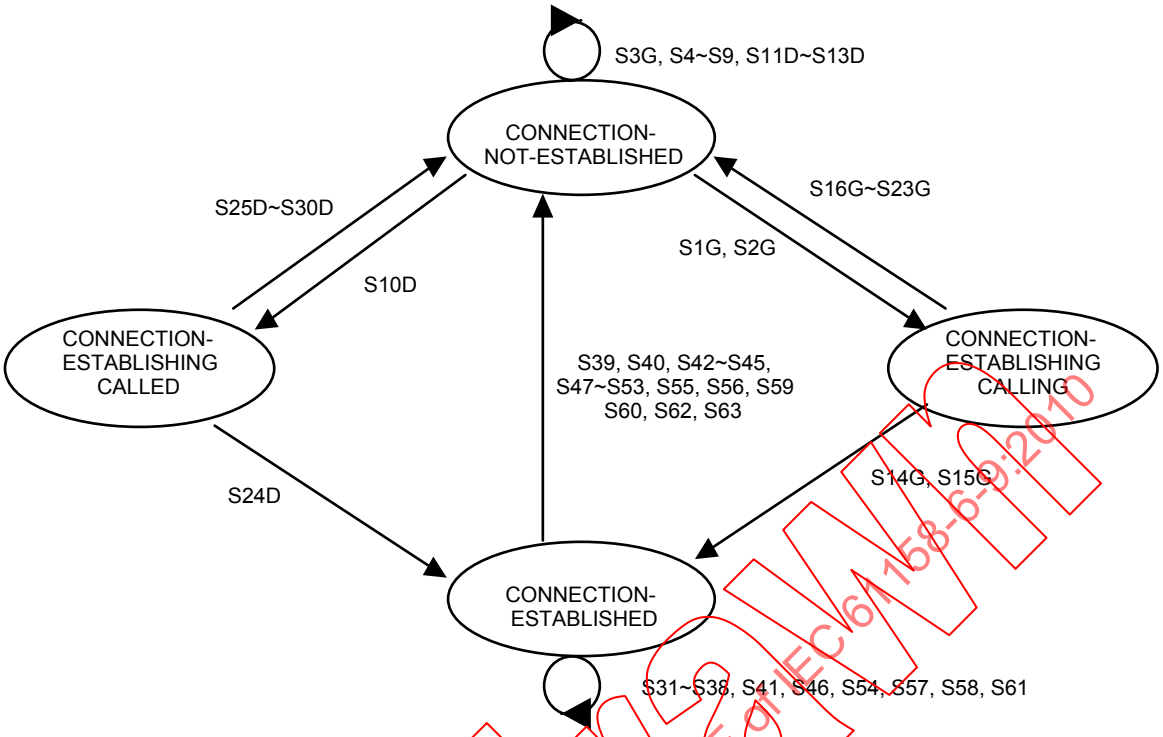


Figure 22 – VCR state machine

NOTE The transaction numbers with “G” suffix are for the calling protocol machine and those with “D” suffix are for the called protocol machine. The transactions without a suffix apply to both protocol machines, or those whose role is yet to be determined.

7.2.2 AP-VCR states

CONNECTION-NOT-ESTABLISHED

The connection is not established. Only the service primitives Initiate.req, ASC.ind, Abort.req and ABT.ind are allowed. All other services shall be rejected with the Abort service.

CONNECTION-ESTABLISHING (CALLING)

The local FMS user wishes to establish the connection. Only the service primitives ASC.cnf(+), ASC.cnf(-), Abort.req and ABT.ind are allowed. All other services shall be rejected with the Abort service. This state is abbreviated as CON-ESTABLISHING-CALLING.

CONNECTION-ESTABLISHING (CALLED)

The remote FMS user wishes to establish the connection. Only the service primitives Initiate.rsp(+), Initiate.rsp(-), Abort.req and ABT.ind are allowed. All other services shall be rejected with the Abort service. This state is abbreviated as CON-ESTABLISHING-CALLED.

CONNECTION-ESTABLISHED

The VCR is established. The service primitives Initiate.req, Initiate.rsp(+), Initiate.rsp(-) are not allowed and shall be rejected with the Abort service. The following actions shall be taken to reset a VCR (Reset VCR).

- Set attribute "Outstanding services Counter Sending" and attribute "Outstanding services Counter Receiving" of the VCR (dynamic part) to 0.
- Set state of the VCR to "CONNECTION-NOT-ESTABLISHED".

7.2.3 AP-VCR initiation state transitions

Table 3 defines the AP-VCR state transitions.

Table 3 – AP-VCR state machine transactions

#	Current state	Event or condition => action	Next state

S1G	CONNECTION-NOT-ESTABLISHED	Initiate.req
		&& VCR is valid
		&& VCR type = QUB
		=>
		ASC.req{
		Data := Initiate-RequestPDU
		}
			CONNECTION – ESTABLISHING-CALLING

S2G	CONNECTION-NOT-ESTABLISHED	Initiate.req
		&& VCR is valid
		&& VCR type <> QUB
		=>
		ASC.req{
		Data := NULL
		}
			CONNECTION – ESTABLISHING-CALLING

S3G	CONNECTION-NOT-ESTABLISHED	Initiate.req
		&& VCR is not valid
		=>
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := VCR error
		}
			CONNECTION-NOT-ESTABLISHED

S4	CONNECTION-NOT-ESTABLISHED	Abort.req => (no actions taken)	CONNECTION-NOT-ESTABLISHED

S5	CONNECTION-NOT-ESTABLISHED	FMS service.primitive other than (nitiate & Abort)
		=>
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := User error
		}
			CONNECTION-NOT-ESTABLISHED

S6	CONNECTION-NOT-ESTABLISHED	ABT.ind => (no actions taken)	CONNECTION-NOT-ESTABLISHED

S7	CONNECTION-NOT-ESTABLISHED	Faulty or unknown FAS service primitive
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		}
			CONNECTION-NOT-ESTABLISHED

S8	CONNECTION-NOT-ESTABLISHED	FAS service primitive other than ASC.ind and ABT.ind
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := Connection state conflict FAS
		}
			CONNECTION-NOT-ESTABLISHED

S9	CONNECTION-NOT-ESTABLISHED	ASC.ind
		&& Data = not allowed, unknown or faulty FMS PDU
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FMS PDU error
		}
			CONNECTION-NOT-ESTABLISHED

S10D	CONNECTION-NOT-ESTABLISHED	ASC.ind
		&& Data Initiate-RequestPDU
		&& VCR is valid
		&& Max FMS PDU length test is positive
		&& FmsFeaturesSupported test is positive
		=>
		Initiate.ind{
		}
			CONNECTION – ESTABLISHING-CALLED

S11D	CONNECTION-NOT-ESTABLISHED	ASC.ind
		&& Data = Initiate-RequestPDU
		&& VCR is invalid
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := VCR error
		}
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state

S12D	CONNECTION-NOT-ESTABLISHED	ASC.ind
		&& Data = Initiate-RequestPDU
		&& VCR is valid
		&& Max FMS PDU length test is negative
		=>
		ASC.rsp(-){
		Data := Initiate-ErrorPDU{
		ErrorCode := max-fms-pdu-size-insufficient
		}
		}
			CONNECTION-NOT-ESTABLISHED

S13D	CONNECTION-NOT-ESTABLISHED	ASC.ind
		&& Data = Initiate-RequestPDU
		&& VCR is valid
		&& Max FMS PDU length test is positive
		&& FmsFeaturesSupported test is negative
		=>
		ASC.rsp(-){
		Data := Initiate-ErrorPDU{
		ErrorCode := feature-not-supported
		}
		}
			CONNECTION-NOT-ESTABLISHED

S14G	CONNECTION - ESTABLISHING -CALLING	ASC.cnf(+)
		&& Data = Initiate-ResponsePDU
		&& VCR type = QUB
		=>
		Initiate.cnf(+){
		}
			CONNECTION-ESTABLISHED

S15G	CONNECTION - ESTABLISHING -CALLING	ASC.cnf(+)
		&& Data = NULL
		&& VCR type = QUU or BNU
		=>
		Initiate.cnf(+){
		}
			CONNECTION-ESTABLISHED

S16G	CONNECTION - ESTABLISHING -CALLING	ASC.cnf(-) && Data = Initiate-ErrorPDU && VCR type = QUB => Initiate.cnf(-){ ErrorCode := Error code in ASC.cnf }, reset VCR	CONNECTION-NOT-ESTABLISHED
S17G	CONNECTION - ESTABLISHING -CALLING	ASC.cnf(-) && Data = NULL && VCR type = QUU or BNU => Initiate.cnf(-){ ErrorCode := Error code in ASC.cnf }, reset VCR	CONNECTION-NOT-ESTABLISHED
S18G	CONNECTION - ESTABLISHING -CALLING	ABT.ind => Abort.ind{ AbortIdentifier := AbortIdentifier of ABT.ind, ReasonCode := ReasonCode of ABT.ind }, reset VCR	CONNECTION-NOT-ESTABLISHED
S19G	CONNECTION - ESTABLISHING -CALLING	Abort.req => ABT.req{ AbortIdentifier := AbortIdentifier of Abort.req, ReasonCode := ReasonCode of Abort.req }, reset VCR	CONNECTION-NOT-ESTABLISHED

S20G	CONNECTION - ESTABLISHING -CALLING	Faulty or unknown FAS service primitive
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state

S21G	CONNECTION - ESTABLISHING -CALLING	FAS service primitive other than ASC.cnf and ABT.ind
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode :=Connection state conflict FAS
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode :=Connection state conflict FAS
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S22G	CONNECTION - ESTABLISHING -CALLING	ASC.cnf
		&& Data = not allowed, unknown or faulty FMS PDU
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FMS PDU error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := FMS PDU error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S23G	CONNECTION - ESTABLISHING -CALLING	FMS service.primitive other than Initiate.rsp and Abort.req
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := User error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := User error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S24D	CONNECTION - ESTABLISHING -CALLED	Initiate.rsp(+)
		=>
		ASC.rsp(+){
		Data = Initiate-ResponsePDU
		}
			CONNECTION-ESTABLISHED

S25D	CONNECTION - ESTABLISHING -CALLED	Initiate.rsp(-)
		=>
		ASC.rsp(-){
		Data = Initiate-ErrorPDU{
		ErrorCode = ErrorCode of Initiate.rsp
		}
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S26D	CONNECTION - ESTABLISHING -CALLED	Abort.req => ABT.req{ AbortIdentifier := AbortIdentifier of Abort.req, ReasonCode := ReasonCode of Abort.req }, reset VCR	CONNECTION-NOT-ESTABLISHED
S27D	CONNECTION - ESTABLISHING -CALLED	ABT.ind => Abort.ind{ AbortIdentifier := AbortIdentifier of ABT.ind, ReasonCode := ReasonCode of ABT.ind }, reset VCR	CONNECTION-NOT-ESTABLISHED

S28D	CONNECTION - ESTABLISHING -CALLED	Faulty or unknown FAS service primitive
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state

S29D	CONNECTION - ESTABLISHING -CALLED	FAS service primitive other ABT.ind
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode :=Connection state conflict FAS
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode :=Connection state conflict FAS
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S30D	CONNECTION - ESTABLISHING -CALLED	FMS service.primitive other than Initiate.rsp and Abort.req
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := User error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := User error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S31	CONNECTION-ESTABLISHED	ConfirmedService.req && OSCS < ActualMaxSCC && InvokeID not existent && PDU length ≤ Max FMS PDU sending && FmsFeaturesSupported test (client) positive => DTC.req{ Data := Confirmed-RequestPDU }, OSCS := OSCS + 1	CONNECTION-ESTABLISHED

S32	CONNECTION-ESTABLISHED	ConfirmedService.req
		&& OSCS = ActualMaxSCC
		=>
		Reject.ind{
		DetectedHere := true,
		Original InvokeID := InvokeID in ConfirmedService.req,
		RejectPDUType := Confirmed-RequestPDU,
		RejectCode := Max-services-overflow
		}
			CONNECTION-ESTABLISHED

S33	CONNECTION-ESTABLISHED	ConfirmedService.req
		&& OSCS < ActualMaxSCC
		&& InvokeID already existent
		=>
		Reject.ind{
		DetectedHere := true,
		Original InvokeID := InvokeID in ConfirmedService.req,
		RejectPDUType := Confirmed-RequestPDU,
		RejectCode := Invoke-ID-exists
		}
			CONNECTION-ESTABLISHED

S34	CONNECTION-ESTABLISHED	ConfirmedService.req
		&& OSCS < ActualMaxSCC
		&& InvokeID not existent
		&& PDU length > Max FMS PDU sending
		=>
		Reject.ind{
		DetectedHere := true,
		Original InvokeID := InvokeID in ConfirmedService.req,
		RejectPDUType := Confirmed-RequestPDU,
		RejectCode := PDU-size
		}
			CONNECTION-ESTABLISHED

S35	CONNECTION-ESTABLISHED	ConfirmedService.req
		&& OSCS < ActualMaxSCC
		&& InvokeID not existent
		&& PDU length ≤ Max FMS PDU sending
		&& FmsFeaturesSupported test (client) negative
		=>
		Reject.ind{
		DetectedHere := true,
		Original InvokeID := InvokeID in ConfirmedService.req,
		RejectPDUType := Confirmed-RequestPDU,
		RejectCode := Feature-not-supported-connection-oriented
		}
			CONNECTION-ESTABLISHED

#	Current state	Event or condition => action	Next state

S36	CONNECTION-ESTABLISHED	UnconfirmedService.req
		&& PDU length ≤ Max FMS PDU sending
		&& FmsFeaturesSupported test (client) positive
		=>
		DTU.req{
		Data := Unconfirmed-PDU
		}
			CONNECTION-ESTABLISHED

S37	CONNECTION-ESTABLISHED	UnconfirmedService.req
		&& PDU length > Max FMS PDU sending
		=>
		Reject.ind{
		DetectedHere := true,
		Original InvokeID := InvokeID in UnconfirmedService.req,
		RejectPDUType := Unconfirmed-PDU,
		RejectCode := PDU-size
		}
			CONNECTION-ESTABLISHED

S38	CONNECTION-ESTABLISHED	UnconfirmedService.req
		&& PDU length ≤ Max FMS PDU sending
		&& FmsFeaturesSupported test (client) negative
		=>
		Reject.ind{
		DetectedHere := true,
		Original InvokeID := InvokeID in UnconfirmedService.req,
		RejectPDUType := Unconfirmed-PDU,
		RejectCode := Feature-not-supported-connection-oriented
		}
			CONNECTION-ESTABLISHED

S39	CONNECTION-ESTABLISHED	Abort.req => ABT.req{ AbortIdentifier := AbortIdentifier of Abort.req, ReasonCode := ReasonCode of Abort.req }, reset VCR	CONNECTION-NOT-ESTABLISHED

S40	CONNECTION-ESTABLISHED	Faulty, unknown or not-allowed FMS service.primitive
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := User error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := User error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S41	CONNECTION-ESTABLISHED	DTC.ind
		&& Data = Confirmed-ServicePDU
		&& PDU length ≤ Max FMS PDU receiving
		&& OSCR < ActualMaxRCC
		&& InvokeID not existent
		&& Features Supported test (server) positive
		=>
		ConfirmedService.ind{
		}
		OSCR := OSCR + 1
			CONNECTION-ESTABLISHED

S42	CONNECTION-ESTABLISHED	DTC.ind
		&& Data = Confirmed-ServicePDU
		&& PDU length > Max FMS PDU receiving
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := PDU-size
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := PDU-size
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state

S43	CONNECTION-ESTABLISHED	DTC.ind
		&& Data = Confirmed-ServicePDU
		&& PDU length ≤ Max FMS PDU receiving
		&& OSCR ≥ ActualMaxRCC
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := Max-services-overflow
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := Max-services-overflow
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S44	CONNECTION-ESTABLISHED	DTC.ind
		&& Data = Confirmed-ServicePDU
		&& PDU length ≤ Max FMS PDU receiving
		&& OSCR < ActualMaxRCC
		&& InvokeID already existent
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-request
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-request
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S45	CONNECTION-ESTABLISHED	DTC.ind
		&& Data = Confirmed-ServicePDU
		&& PDU length ≤ Max FMS PDU receiving
		&& OSCR < ActualMaxRCC
		&& InvokeID not existent
		&& Features Supported test (server) negative
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := Feature-not-supported
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := Feature-not-supported
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S46	CONNECTION-ESTABLISHED	DTU.ind
		&& Data = Unconfirmed-PDU
		&& PDU length ≤ Max FMS PDU receiving
		&& Features Supported test (server) positive
		=>
		UnconfirmedService.ind{
		}
			CONNECTION-ESTABLISHED

S47	CONNECTION-ESTABLISHED	DTU.ind
		&& Data = Unconfirmed-PDU
		&& PDU length > Max FMS PDU receiving
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := PDU-size
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := PDU-size
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state

S48	CONNECTION-ESTABLISHED	DTU.ind
		&& Data = Unconfirmed-PDU
		&& PDU length ≤ Max FMS PDU receiving
		&& Features Supported test (server) negative
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := Feature-not-supported
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := Feature-not-supported
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S49	CONNECTION-ESTABLISHED	ABT.ind => Abort.ind{ AbortIdentifier := AbortIdentifier of ABT.ind, ReasonCode := ReasonCode of ABT.ind }, reset VCR	CONNECTION-NOT-ESTABLISHED

S50	CONNECTION-ESTABLISHED	ASC.ind
		&& Data = Initiate-RequestPDU
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := Connection-state-conflict-FMS
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := Connection-state-conflict-FMS
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S51	CONNECTION-ESTABLISHED	Faulty or unknown FAS service primitive
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := FAS error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S52	CONNECTION-ESTABLISHED	Not-allowed FAS service primitive
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := Connection-state-conflict-FMS
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := Connection-state-conflict-FMS
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S53	CONNECTION-ESTABLISHED	valid FAS service primitive
		&& Data = not-allowed, unknown or faulty FMS PDU
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FMS-PDU-error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := FMS-PDU-error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state
S54	CONNECTION-ESTABLISHED	ConfirmedService.rsp && InvokeID (server) existent && services in .rsp and .ind are identical && PDU length ≤ Max FMS PDU sending => DTC.rsp{ Data := Confirmed-ResponsePDU }, OSCR := OSCR - 1	CONNECTION-ESTABLISHED

S55	CONNECTION-ESTABLISHED	ConfirmedService.rsp
		&& InvokeID (server) not existent
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-response
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-response
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S56	CONNECTION-ESTABLISHED	ConfirmedService.rsp
		&& InvokeID (server) existent
		&& services in .rsp and .ind are not identical
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := service-error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := service-error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S57	CONNECTION-ESTABLISHED	ConfirmedService.rsp
		&& InvokeID (server) existent
		&& services in .rsp and .ind are identical
		&& PDU length > Max FMS PDU sending
		=>
		DTC.rsp{
		Data := Reject-PDU{
		Original InvokeID := InvokeID in ConfirmedService.req,
		RejectCode := PDU-size
		}
		},
		OSCR := OSCR - 1
			CONNECTION-ESTABLISHED

S58	CONNECTION-ESTABLISHED	DTC.cnf && Data = Confirmed-ResponsePDU && PDU length ≤ Max FMS PDU receiving && InvokeID (client) existent && services in .cnf and .req are identical => ConfirmedService.cnf{ }, OSCR := OSCR	CONNECTION-ESTABLISHED

S59	CONNECTION-ESTABLISHED	DTC.cnf
		&& Data = Confirmed-ResponsePDU
		&& PDU length > Max FMS PDU receiving
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := PDU-size
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := PDU-size
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

#	Current state	Event or condition => action	Next state

S60	CONNECTION-ESTABLISHED	DTC.cnf
		&& Data = Confirmed-ResponsePDU
		&& PDU length ≤ Max FMS PDU receiving
		&& InvokeID (client) not existent
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-response
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-response
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S61	CONNECTION-ESTABLISHED	DTC.cnf && Data = Reject-PDU && Original InvokeID (client) existent && RejectCode:= PDU-size => Reject.ind{ DetectedHere := false, Original InvokeID := InvokeID in Reject-PDU, RejectPDUType := Confirmed-Response-PDU, RejectCode := PDU-size }, OSCS := OSCS -1	CONNECTION-ESTABLISHED

S62	CONNECTION-ESTABLISHED	DTC.cnf
		&& Data = Reject-PDU
		&& Original InvokeID (client) not existent
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-response
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := InvokeID-error-response
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

S63	CONNECTION-ESTABLISHED	DTC.cnf
		&& Data = Reject-PDU
		&& Original InvokeID (client) existent
		&& RejectCode:<> PDU-size
		=>
		ABT.req{
		AbortIdentifier := FMS,
		ReasonCode := FMS-PDU-error
		},
		Abort.ind{
		AbortIdentifier := FMS,
		ReasonCode := FMS-PDU-error
		},
		reset VCR
			CONNECTION-NOT-ESTABLISHED

7.2.4 Primitives exchanged between FAL-User and VCR PM

See Table 4 and Table 5 for the primitives exchanged between the FAL-User and the VCR PM.

Table 4 – Primitives issued by FAL-User to VCR PM

Primitive name	Source	Associated parameters and functions
Terminate.req	FAL-User	Refer to FAL service Definition (IEC 61158-5 subseries)
Initiate.req	FAL-User	
Initiate.rsp(+)	FAL-User	
Initiate.rsp(-)	FAL-User	
ConfirmedService.req	FAL-User	
ConfirmedService.rsp	FAL-User	
UnconfirmedService.req	FAL-User	

Table 5 – Primitives issued by VCR PM to FAL-User

Primitive name	Source	Associated parameters and functions
Terminate.ind	VCR PM	Refer to FAL service Definition (IEC 61158-5 subseries)
Initiate.ind	VCR PM	
Initiate.cnf(+)	VCR PM	
Initiate.cnf(-)	VCR PM	
ConfirmedService.ind	VCR PM	
ConfirmedService.cnf	VCR PM	
UnconfirmedService.ind	VCR PM	

7.2.5 Primitives exchanged between FSPM and the VCR PM

Table 6 and Table 7 define the primitives used by the FSPM

Table 6 – Primitives issued by VCR PM to FSPM

Primitive name	Source	Associated parameters	Functions
ASC.req	VCR PM	Arep_Id, Data, Remote_DLCEP_Address	This primitive is used to convey an Associate request primitive from the VCR PM to the FSPM.
ASC.rsp(+)	VCR PM	Arep_Id, Data	This primitive is used to convey an Associate response(+) primitive from the VCR PM to the FSPM.
ASC.rsp(-)	VCR PM	Arep_Id, Data	This primitive is used to convey an Associate response(-) primitive from the VCR PM to the FSPM.
Abort.req	VCR PM	Arep_Id, Identifier, Reason_Code, Additional_Detail	This primitive is used to convey an Abort request primitive from the VCR PM to the FSPM.
CS.req	VCR PM	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) request primitive from the VCR PM to the FSPM.
CS.rsp	VCR PM	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) response primitive from the VCR PM to the FSPM.
UCS.req	VCR PM	Arep_Id, Remote_DLSAP_Address, Data	This primitive is used to convey an Unconfirmed Send (UCS) request primitive from the VCR PM to the FSPM.
FCMP.req	VCR PM	Arep_Id	This primitive is used to convey an FAL-Compel (FCMP) request primitive from the VCR PM to the FSPM.
GBM.req	VCR PM	Arep_Id	This primitive is used to convey a Get-Buffered-Message (GBM) request primitive from the VCR PM to the FSPM.

Table 7 – Primitives issued by FSPM to VCR PM

Primitive name	Source	Associated parameters	Functions
ASC.ind	FSPM	Arep_Id, Data	This primitive is used to convey an Associate indication primitive from the FSPM to the VCR PM.
ASC.cnf(+)	FSPM	Arep_Id, Data	This primitive is used to convey an Associate result(+) primitive from the FSPM to the VCR PM.
ASC.cnf(-)	FSPM	Arep_Id, Data	This primitive is used to convey an Associate result(-) primitive from the FSPM to the VCR PM.
Abort.ind	FSPM	Arep_Id, Locally_Generated, Identifier, Reason_Code, Additional_Detail	This primitive is used to convey an Abort indication primitive from the FSPM to the VCR PM.
CS.ind	FSPM	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) indication primitive from the FSPM to the VCR PM.
CS.cnf	FSPM	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) confirmation primitive from the FSPM to the VCR PM.
UCS.ind	FSPM	Arep_Id, Remote_DLSAP_Address, Duplicate_FAL-SDU, Data, Local_Timeliness, Remote_Timeliness	This primitive is used to convey an Unconfirmed Send (UCS) indication primitive from the FSPM to the VCR PM.
FCMP.cnf	FSPM	Arep_Id, Status	This primitive is used to convey an FAL-Compel (FCMP) confirmation primitive from the FSPM to the VCR PM.
GBM.cnf(+)	FSPM	Arep_Id, Duplicate_FAL-SDU, Data, Local_Timeliness, Remote_Timeliness	This primitive is used to convey a Get-Buffered-Message (GBM) positive confirmation primitive from the FSPM to the VCR PM.
GBM.cnf(-)	FSPM	Arep_Id	This primitive is used to convey a Get-Buffered-Message (GBM) negative confirmation primitive from the FSPM to the VCR PM.
FSTS.ind	FSPM	Arep_Id, Reported_Status	This primitive is used to convey an FAL-Status (FSTS) indication primitive from the FSPM to the VCR PM.

8 FAL service protocol machine (FSPM)

8.1 General

FAS service Protocol Machine is common to all the AREP types. Only applicable primitives are different among different AREP types. It has one state called "ACTIVE" as shown in Figure 23.

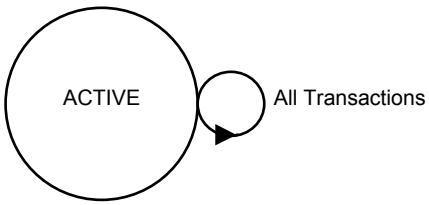


Figure 23 – State transition diagram of FSPM

8.2 FSPM state tables

Table 8 and Table 9 specifiy the FSPM protocol machine.

Table 8 – FSPM state table – sender transactions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	ASC.req
		&& SelectArep (Arep_Id) = "True"
		=>
		ASC_req {
		user_data := Data,
		remote_dlcep_address := Remote_DLCEP_Address
		}
			ACTIVE

S2	ACTIVE	ASC.rsp(+)
		&& SelectArep (Arep_Id) = "True"
		=>
		ASC_rsp(+) {
		user_data := Data
		}
			ACTIVE

S3	ACTIVE	ASC.rsp(-)
		&& SelectArep (Arep_Id) = "True"
		=>
		ASC_rsp(-) {
		user_data := Data
		}
			ACTIVE

S4	ACTIVE	DTU.req
		&& SelectArep (Arep_Id) = "True"
		=>
		DTU_req {
		remote_dlsap_address := Remote_DLSAP_Address,
		user_data := Data
		}
			ACTIVE

S5	ACTIVE	DTC.req
		&& SelectArep (Arep_Id) = "True"
		=>
		DTC_req {
		user_data := Data
		}
			ACTIVE

S6	ACTIVE	DTC.rsp
		&& SelectArep (Arep_Id) = "True"
		=>
		DTC_rsp {
		user_data := Data
		}
			ACTIVE

S7	ACTIVE	Abort.req
		&& SelectArep (Arep_Id) = "True"
		=>
		Abort_req {
		identifier := Identifier,
		reason_code := Reason_Code,
		additional_detail := Additional_Detail
		}
			ACTIVE

S8	ACTIVE	FCMP.req
		&& SelectArep (Arep_Id) = “True”
		=>
		FCMP_req {
		}
			ACTIVE

S9	ACTIVE	GBM.req
		&& SelectArep (Arep_Id) = “True”
		=>
		GBM_req {
		}
			ACTIVE

NOTE 1 A primitive generated in the FSPM sender state machine is sent to an appropriate ARPM that is selected by the FSPM using the SelectArep function. The Arep_Id parameter supplied by the FAS user is the argument of this function.

NOTE 2 If the SelectArep function returns the value of False, it is a local matter to report such instance and the FSPM does not generate any primitives to the ARPM.

Table 9 – FSPM state table – receiver transactions

#	Current state	Event or condition => action	Next state

R1	ACTIVE	ASC_ind
		=>
		ASC.ind {
		Arep_Id := arep_id,
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	ASC_cnf(+)
		=>
		ASC.cnf(+) {
		Arep_Id := arep_id,
		Data := user_data
		}
			ACTIVE

R3	ACTIVE	ASC_cnf(-)
		=>
		ASC.cnf(-) {
		Arep_Id := arep_id,
		Data := user_data
		}
			ACTIVE

R4	ACTIVE	DTU_ind
		=>
		DTU.ind {
		Arep_Id := arep_id,
		Remote_DLSAP_Address := remote_dlsap_address,
		Duplicate_FAS-SDU := duplicate_fas_sdu,
		Data := user_data,
		Local_Timeliness := local_timeliness,
		Remote_Timeliness := remote_timeliness
		}
			ACTIVE

R5	ACTIVE	DTC_ind
		=>
		DTC.ind {
		Arep_Id := arep_id,
		Data := user_data
		}
			ACTIVE

R6	ACTIVE	DTC_cnf
		=>
		DTC.cnf {
		Arep_Id := arep_id,
		Data := user_data
		}
			ACTIVE

R7	ACTIVE	Abort_ind
		=>
		Abort.ind {
		Arep_Id := arep_id,
		Locally_Generated := locally_generated,
		Identifier := identifier,
		Reason_Code := reason_code,
		Additional_Detail := additional_detail
		}
			ACTIVE

R8	ACTIVE	FCMP_cnf
		=>
		FCMP.cnf {
		Arep_Id := arep_id,
		Status := status
		}
			ACTIVE

R9	ACTIVE	GBM_cnf(+)
		=>
		GBM.cnf(+) {
		Arep_Id := arep_id,
		Duplicate_FAS-SDU := duplicate_fas_sdu,
		Data := user_data,
		Local_Timeliness := local_timeliness,
		Remote_Timeliness := remote_timeliness
		}
			ACTIVE

R10	ACTIVE	GBM_cnf(-)
		=>
		GBM.cnf(-) {
		Arep_Id := arep_id,
		}
			ACTIVE

#	Current state	Event or condition => action	Next state

R11	ACTIVE	FSTS_ind
		=>
		FSTS.ind {
		Arep_Id := arep_id,
		Reported_Status := reported_status
		}
			ACTIVE

8.3 Functions used by FSPM

Table 10 defines the function used by the FSPM

Table 10 – Function SelectArep()

Name	SelectArep	Used in	FSPM
Input		Output	
Arep_Id		True || False	
Function			
Looks for the AREP entry that is specified by the Arep_Id parameter. The Arep_Id parameter is provided with the FAS user service primitives.			

8.4 Parameters of FSPM/ARPM primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 11.

Table 11 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
arep_id	This parameter is used to unanimously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this specification.
user_data	This parameter conveys FAS-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fas_sdu	This parameter conveys value that is used for the Duplicate_FAS-SDU parameter.
remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Like Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

9 Application relationship protocol machines (ARPMs)

9.1 AREP mapping to data-link layer

9.1.1 General

This section describes the mapping of the FAS to the Fieldbus data-link layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the data-link layer specification; rather, it defines how they are used by each of the AR classes. A means to configure and monitor the values of these attributes is provided by Network Management.

The following class definitions describe the DLSAP attributes and the DLME attributes required to support each of the AREP classes.

NOTE Undefined attributes use the same definitions as those previously defined.

9.1.2 DLL mapping of QUU AREP class

9.1.2.1 QUU AREP class formal model

The DLL Mapping attributes and their permitted values and the DLL services used with the QUU AREP class are defined in this subclause.

CLASS: **QuuCl**
PARENT CLASS: QueuedUser-TriggeredUnidirectionalAREP
ATTRIBUTES:

1.	(m)	KeyAttribute:	LocalDlsapAddress
2.	(m)	Attribute:	RemoteDlsapAddress
3.	(m)	Attribute:	LocalDlsapRole (Basic, Group)
4.	(c)	Constraint:	Role = Source
4.1	(m)	Attribute:	DefaultQosAsSender
4.2	(m)	Attribute	DllPriority (Urgent, Normal, TimeAvailable)
4.3	(m)	Attribute:	MaxConfirmDelayOnUnitdata
4.4	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
4.5	(m)	Attribute:	DlSchedulingPolicy (Implicit)
4.6	(m)	Attribute:	ExplicitQueue (True, False)
5.	(c)	Constraint:	ExplicitQueue = True
5.1	(m)	Attribute:	QueueBindings—either for a sender or a receiver
5.2	(m)	Attribute:	QueueIdentifier
5.3	(m)	Attribute:	MaxQueueDepth
5.4	(m)	Attribute:	MaxDlsduSize

DLL SERVICES:

1.	(m)	OpsService:	DL-Unitdata
2.	(c)	Constraint:	ExplicitQueue = True
3.	(m)	OpsService:	DL-Get

9.1.2.2 Attributes

9.1.2.2.1 LocalDlsapAddress

This attribute specifies the DLSAP address to which this AREP is attached. This attribute is a DLSAP-address if the Role attribute has the value of Source, and either a group DL-address or a DLSAP-address if the Role attribute has the value of Sink.

This attribute supplies the value for the “DL(SAP)-address” parameter specified in the DLL.

This attribute contains the following three sub-attributes: Link Address, Node Address, and Selector.

9.1.2.2.2 RemoteDlsapAddress

This attribute specifies the remote address to which FAS-PDUs are sent (for Source AREPs), or from which they are received (for Sink AREPs).

If the ConfigurationType attribute is Linked, the value of this attribute has been configured. If it is Free, the value of this attribute is provided with a service request.

This attribute contains the following three sub-attributes: Link Address, Node Address, and Selector.

9.1.2.2.3 LocalDlsapRole

This attribute specifies the behavior of the local DLSAP to be used. If the Role is Source, this attribute has the value of Basic. If the Role is Sink, it has the value of either Basic or Group.

This attribute supplies the value for the “DL(SAP)-role” parameter specified by the DLL.

9.1.2.2.4 DefaultQosAsSender

The DefaultQosAsSender attributes specify the DLL quality of service that is used by the sending AREP. The receiving DLL shall support the quality of service specified by these attributes.

9.1.2.2.5 DllPriority

This attribute defines the DLL priority, and thus restricts the maximum length of an FAS-PDU, of the conveyance path of an AR.

This attribute supplies the value for the “DLL priority” parameter of the DLL. The values Urgent, Normal, and Time-Available correspond to URGENT, NORMAL, and TIME-AVAILABLE as defined in IEC 61158-3-1 and IEC 61158-4-1, respectively.

NOTE It is not possible to use different priorities for each FAS-PDU sent from the same QUU AREP.

9.1.2.2.6 MaxConfirmDelayOnUnitdata

This attribute specifies the maximum confirmation delay for a local confirmation from a DL-Unitdata request primitive.

This attribute supplies the value for the “Max confirm delay on locally-confirmed DL-Unitdata” parameter of the DLL.

9.1.2.2.7 DlpduAuthentication

This attribute specifies the lower bound of the length of the DL-addresses to be used by the DLL.

This attribute supplies the value for the “DLPDU authentication” parameter of the DLL. The values Ordinary, Source, and Maximal correspond to ORDINARY, SOURCE, and MAXIMAL as defined in IEC 61158-3-1 and IEC 61158-4-1, respectively.

9.1.2.2.8 DlSchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAS-PDU as soon as it is passed from the FAS.

This attribute supplies the value for DL-Scheduling-policy attribute. Only the value Implicit is used. It corresponds to IMPLICIT as defined in IEC 61158-3-1 and IEC 61158-4-1.

9.1.2.2.9 ExplicitQueue

This attribute specifies, when True, that the characteristics of the associated sending and receiving queues are explicitly configured and managed through the Network Management. The value of False means that queues with implementation specific depth and length are provided by the DLL.

9.1.2.2.10 QueueBindings

The following attributes specify the explicit queue that is bound to this DLSAP. For a sender, the queue is for sending. For a receiver, it is for receiving.

9.1.2.2.11 QueueIdentifier

This attribute provides a local means to identify a queue that is associated with this DLSAP.

This attribute supplies the value for the “Buffer-or-queue-identifier” parameter defined in the DLL.

9.1.2.2.12 MaxQueueDepth

This attribute specifies the maximum number of FAS-PDUs that can be queued for sending or receiving.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL.

9.1.2.2.13 MaxDlsduSize

This attribute specifies the maximum length of an FAS-PDU that can be sent or received by the DLL.

This attribute supplies the value for the “Maximum DLSDU size” parameter of the DLL.

9.1.2.3 DLL services

Refer to IEC 61158-3-1 for DLL service descriptions.

9.1.3 DLL mapping of QUB AREP class

9.1.3.1 QUB AREP class formal model

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB AREP class are defined in this subclause.

CLASS: **QubCo**
PARENT CLASS: **QueuedUser-TriggeredBidirectionalAREP**
ATTRIBUTES:

1.	(m)	KeyAttribute:	LocalDlcepAddress
2.	(m)	Attribute:	RemoteDlcepAddress
3.	(m)	KeyAttribute:	DlcepDlIdentifier
4.	(m)	Attribute:	DlsapRole (Basic)
5.	(m)	Attribute:	QosParameterSet
5.1	(m)	Attribute:	DlcepClass (Peer)
5.2	(m)	Attribute:	DlcepDataDeliveryFeatures
5.2.1	(m)	Attribute:	FromRequesterToResponder (Classical, Disordered)
5.2.2	(m)	Attribute:	FromResponderToRequester (Classical, Disordered)
5.3	(m)	Attribute:	Priority
5.3.1	(m)	Attribute:	DlPriority (Urgent, Normal, TimeAvailable)
5.3.2	(m)	Attribute:	DlPriorityNegotiated (Urgent, Normal, TimeAvailable)
5.4	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
5.5	(m)	Attribute:	ResidualActivity
5.5.1	(m)	Attribute:	ResidualActivityAsSender (True, False)
5.5.2	(m)	Attribute:	ResidualActivityAsReceiver (True, False)
5.6	(m)	Attribute:	MaxConfirmDelay
5.6.1	(m)	Attribute:	MaxConfirmDelayOnDlConnect
5.6.2	(m)	Attribute:	MaxConfirmDelayOnDlData
5.7	(m)	Attribute:	DlSchedulingPolicy (Implicit)
5.8	(m)	Attribute:	ExplicitQueue (True, False)
5.9	(c)	Constraint:	ExplicitQueue = True
5.9.1	(m)	Attribute:	MaxDlsduSizes
5.9.2	(m)	Attribute:	MaxDlsduSizeFromRequester
5.9.3	(m)	Attribute:	MaxDlsduSizeFromResponder
5.9.4	(m)	Attribute:	MaxDlsduSizeFromRequesterNegotiated
5.9.5	(m)	Attribute:	MaxDlsduSizeFromResponderNegotiated
5.10	(m)	Attribute:	QueueBindings
5.10.1	(m)	Attribute:	SendingBufferOrQueueIdentifier
5.10.2	(m)	Attribute:	ReceivingBufferOrQueueIdentifier
5.11	(m)	Attribute:	MaxQueueDepth
5.11.1	(m)	Attribute:	MaxSendingQueueDepth
5.11.2	(m)	Attribute:	MaximimReceivingQueueDepth

DLL SERVICES:

1.	(m)	OpsService:	DL-Data
2.	(c)	Constraint:	ExplicitQueue = True
2.1	(m)	OpsService:	DL-Get
3.	(m)	OpsService:	DL-Connect
4.	(m)	OpsService:	DL-Connection-Established
5.	(m)	OpsService:	DL-Disconnect

9.1.3.2 Attributes

9.1.3.2.1 LocalDlcepAddress

This attribute specifies the local DLCEP address and identifies the DLCEP.

The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

NOTE The value of this attribute is also carried in the Associate Request PDU.

This attribute contains the following three sub-attributes: Link Address, Node Address, and Selector.

NOTE Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDlsapAddress attribute is a configurable attribute of the FAS.

9.1.3.2.2 RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP.

This attribute supplies the value for called DLCEP-address of the DL-Connect service.

NOTE The value of this attribute is also carried in the header part of the Associate Request PDU.

This attribute contains the following three sub-attributes: Link Address, Node Address, and Selector.

9.1.3.2.3 DlcepDlIdentifier

The DlcepDlIdentifier attribute identifies the DLCEP.

This attribute supplies the value for the DLCEP DL-identifier parameter of the DL-Connect service. It is returned to the calling FAS by the DL-Connect request primitive, and by the DL-Connect indication primitive to the called FAS.

9.1.3.2.4 DlsapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP.

This attribute supplies the value for the “DL(SAP)-role” parameter. The possible value Basic corresponds to BASIC as defined in the data-link layer specification.

9.1.3.2.5 QosParameterSet

The QosParameterSet attributes specify the DL quality of service that is used by this AREP. These attribute values may be negotiated with the remote AREP.

9.1.3.2.6 DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP.

This attribute supplies the value for the “DLCEP class” parameter of the DLL. The possible value of this attribute is Peer and corresponds to Peer defined by the DLL.

9.1.3.2.7 DlcepDataDeliveryFeatures

These two attributes specify data delivery features of the DLL.

The permitted values Classical and Disordered correspond, respectively to CLASSICAL and DISORDERED defined by the data-link layer specification.

NOTE The FromRequesterToResponder and FromResponderToRequester attributes shall have the same value.

9.1.3.2.8 FromRequesterToResponder

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP. It supplies the value for the “DLCEP data delivery features from requester to responder(s)” parameter defined in the DLL.

9.1.3.2.9 FromResponderToRequester

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP. It supplies the value for the “DLCEP data delivery features from responder(s) to requester” parameter defined in the DLL.

9.1.3.2.10 DllPriority

This attribute specifies the configured value of the DLL priority.

NOTE It is not possible to use different priorities for each FAS-PDU sent from the same QUB AREP. Also, it is not permitted to use different priorities for the send and receive conveyance paths.

9.1.3.2.11 DllPriorityNegotiated

This attribute specifies the negotiated value of the DLL priority.

9.1.3.2.12 DlpduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the “DLPDU-authentication” parameter of the DLL. The permitted value Ordinary, Source, and Maximal correspond to ORDINARY, Source, and MAXIMAL, respectively, as defined in IEC 61158-3-1 and IEC 61158-4-1 of this International Standard.

9.1.3.2.13 ResidualActivityAsSender

This attribute specifies sender’s DLC residual activity. It supplies the value for the “Residual activity as sender” parameter defined in the DLL. The possible values are “True” and “False.”

9.1.3.2.14 ResidualActivityAsReceiver

This attribute specifies receiver’s DLC residual activity. It supplies the value for the “Residual activity as receiver” parameter defined in the DLL. The possible values are “True” and “False.”

9.1.3.2.15 MaxConfirmDelay

This attribute specifies the maximum confirmation delay of certain DLL connection-oriented services.

9.1.3.2.16 MaxConfirmDelayOnDlConnect

This attribute species the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query” parameter specified in the data-link layer specification.

9.1.3.2.17 MaxConfirmDelayOnDlData

This attribute species the maximum confirmation delay for a confirmation from a DL-Data service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Data” parameter specified in IEC 61158-3-1 and IEC 61158-4-1 of this International Standard.

9.1.3.2.18 DlSchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAS-PDU as soon as possible.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Implicit corresponds to IMPLICIT defined in IEC 61158-3-1 and IEC 61158-4-1 of this International Standard.

9.1.3.2.19 ExplicitQueue

9.1.3.2.20 MaxDlsduSizeFromRequester

This attribute specifies the configured value of the maximum length of an FAS-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requester” parameter of the DLL.

9.1.3.2.21 MaxDlsduSizeFromResponder

This attribute specifies the configured value of the maximum length of an FAS-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from responder” parameter of the DLL.

9.1.3.2.22 MaxDlsduSizeFromRequesterNegotiated

This attribute specifies the negotiated value of the maximum length of an FAS-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

9.1.3.2.23 MaxDlsduSizeFromResponderNegotiated

This attribute specifies the negotiated value of the maximum length of an FAS-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

9.1.3.2.24 SendingBufferOrQueueIdentifier

This attribute provides a local means to identify a queue that is used to store sending FAS-PDUs.

This attribute supplies the value for the “Buffer-and-queue bindings as sender” parameter of the DLL.

9.1.3.2.25 ReceivingBufferOrQueueIdentifier

This attribute provides a local means to identify a queue that is used to store receiving FAS-PDUs.

This attribute supplies the value for the “Buffer-and-queue bindings as receiver” parameter of the DLL.

9.1.3.2.26 MaxSendingQueueDepth

This attribute specifies the maximum number of FAS-PDUs that can be queued for transmission.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL for Send queues.

9.1.3.2.27 MaxReceivingQueueDepth

This attribute specifies the maximum number of FAS-PDUs that can be queued at reception.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL for Receive queues.

9.1.3.3 DLL services

Refer to IEC 61158-3-1 for DLL service descriptions.

9.1.4 DLL mapping of BNU AREP class

9.1.4.1 BNU AREP class formal model

The DLL Mapping attributes and their permitted values and the DLL services used with the BNU AREP class are defined in this subclause.

CLASS: **BnuCo**
PARENT CLASS: **BufferedNetworkScheduledUnidirectionalAREP**
ATTRIBUTES:

1.	(m)	KeyAttribute:	PublisherDlcepAddress
2.	(m)	KeyAttribute:	DlcepDlIdentifier
3.	(m)	Attribute:	DlsapRole (Basic)
4.	(m)	Attribute:	QosParameterSet
4.1	(m)	Attribute:	DlcepClass (Publisher, Subscriber)
4.2	(m)	Attribute:	DllPriority (Urgent, Normal, TimeAvailable)
4.3	(m)	Attribute:	DlpduAuthentication (Ordinary, Source)
4.4	(m)	Attribute:	ResidualActivity
4.4.1	(m)	Attribute:	AsSender (False)
4.5	(m)	Attribute:	MaxConfirmDelayOnDlConnect
4.6	(m)	Attribute:	DlSchedulingPolicy (Explicit)
4.7	(m)	Constraint:	DlcepClass = Publisher
4.7.1	(m)	Attribute:	FromRequesterToResponder (Ordered, Unordered)
4.7.2	(m)	Attribute:	MaxDlsduSizeFromRequester
4.7.3	(m)	Attribute:	SendingBufferOrQueueIdentifier
4.7.4	(o)	Attribute:	SenderTimeliness
4.7.4.1	(o)	Attribute:	PublisherDlTimelinessClass (Residence, Update, Synchronous, None)
4.7.4.2	(o)	Attribute:	PublisherTimeWindowSize
4.7.4.3	(o)	Constraint:	PublisherDlTimelinessClass == Update || Synchronous
4.7.4.3.1	(o)	Attribute:	PublisherSynchronizingDlcep
4.8	(m)	Constraint:	DlcepClass = Subscriber
4.8.1	(m)	Attribute:	FromResponderToRequester (Ordered, Unordered)

4.8.2	(m)	Attribute:	MaxDlsduSizeFromResponder	
4.8.3	(m)	Attribute	ReceivingBufferOrQueueIdentifier	
4.8.4	(o)	Attribute:	ReceiverTimeliness	
4.8.4.1	(o)	Attribute:	PublisherDlTimelinessClass (Residence, Update, Synchronous, None)	
4.8.4.2	(o)	Attribute:	PublisherTimeWindowSize	
4.8.4.3	(o)	Attribute:	SubscriberDlTimelinessClass (Residence, Update, Synchronous, None)	
4.8.4.4	(o)	Attribute:	SubscriberTimeWindowSize	
4.8.4.5	(o)	Constraint:	SubscriberDlTimelinessClass == Update || Synchronous	
4.8.4.5.1	(o)	Attribute:	SubscriberSynchronizingDlcep	
5.	(m)	Attribute:	LasScheduled	(True, False)

DLL SERVICES:

1.	(m)	OpsService:	DL-Put
2.	(m)	OpsService:	DL-Get
3.	(m)	OpsService:	DL-Connect
4.	(m)	OpsService:	DL-Connection-Established
5.	(m)	OpsService:	DL-Disconnect
6.	(m)	OpsService:	DL-Buffer-Received
7.	(m)	OpsService:	DL-Buffer-Sent
8.	(m)	Constraint:	LasScheduled == False
9.	(m)	OpsService:	DL-Compel-Service

9.1.4.2 Attributes

9.1.4.2.1 PublisherDlcepAddress

This attribute specifies the Publisher's DLCEP address and identifies the DLCEP.

The value of this attribute is used as the "DLCEP-address" parameter of the DLL.

This attribute contains the following three sub-attributes: Link Address, Node Address, and Selector.

NOTE Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDlsapAddress attribute is a configurable attribute of the FAS.

9.1.4.2.2 DlcepDlIdentifier

9.1.4.2.3 DlsapRole

9.1.4.2.4 QosParameterSet

9.1.4.2.5 DlcepClass

9.1.4.2.6 DllPriority

NOTE It is not possible to use different priorities for each FAS-PDU sent from the same BNU AREP.

9.1.4.2.7 DlpduAuthentication

9.1.4.2.8 ResidualActivity

9.1.4.2.9 MaxConfirmDelayOnDlConnect

9.1.4.2.10 DlSchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAS-PDU when it is instructed to do so by a stimulus from the network.

This attribute supplies the value for the "DL-Scheduling-policy" parameter of the DLL. The permitted value Explicit corresponds to EXPLICIT defined IEC 61158-3-1 and IEC 61158-4-1 of this International Standard.

9.1.4.2.11 FromRequesterToResponder

This attribute is used if the Role attribute has a value of “Publisher” and specifies the DLL data delivery feature of the AREP.

It supplies the value for the “DLCEP data delivery features from requester to responder(s)” parameter of the DLL. The possible values Ordered and Unordered correspond to ORDERED and UNORDERED defined in IEC 61158-3-1 and IEC 61158-4-1 of this International Standard.

If the DuplicatePduDetectionSupported attribute is True, this attribute has the value of “Ordered.”

9.1.4.2.12 MaxDlsduSizeFromRequester

This attribute is used if the Role attribute has a value of “Publisher” and specifies the maximum length of an FAS-PDU that can be sent from this AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requester” parameter of the DLL.

9.1.4.2.13 SendingBufferOrQueueIdentifier

This attribute provides a local means to identify a buffer that is used to store FAS-PDUs for sending. This attribute supplies the value for the “Buffer-and-queue bindings as sender” parameter of the DLL.

9.1.4.2.14 FromResponderToRequester

This attribute is used if the Role attribute has a value of “Subscriber” and specifies the DLL data delivery feature of the AREP.

It supplies the value for the “DLCEP data delivery features from responder(s) to requester” parameter of the DLL. The possible values Ordered and Unordered correspond to ORDERED and UNORDERED defined in the data-link layer specification.

If the DuplicatePduDetectionSupported attribute is True, this attribute has the value of “Ordered.”

9.1.4.2.15 MaxDlsduSizeFromResponder

This attribute is used if the Role attribute has a value of “Subscriber” and specifies the maximum length of an FAS-PDU that can be received by this AREP.

This attribute supplies the value for the “Maximum DLSDU size from responder” parameter of the DLL.

9.1.4.2.16 ReceivingBufferOrQueueIdentifier

This attribute provides a local means to identify a buffer that is used to store FAS-PDUs for receiving. This attribute supplies the value for the “Buffer-and-queue bindings as receiver” parameter of the DLL.

9.1.4.2.17 SenderTimeliness

9.1.4.2.18 PublisherDlTimelinessClass

This optional attribute provides the timeliness class of a Publisher provided by the DLL. This attribute supplies the value for the “DL-timeliness-class” parameter of the DLL. The permitted

values Residence, Update, Synchronous, and None correspond to RESIDENCE, UPDATE, SYNCHRONOUS, and NONE defined by the DLL.

9.1.4.2.19 PublisherTimeWindowSize

This optional attribute provides time window of a Publisher provided by the DLL. This attribute supplies the value for the “Time window size” parameter of the DLL.

9.1.4.2.20 PublisherSynchronizingDlcep

This optional attribute is present when the PublisherDlTimelinessClass attribute has the value of Update or Synchronous and provides a DLCEP that is used to generate synchronizing events by the DLL. The FAS user may derive the AREP from which the synchronizing events are delivered by this attribute value. This attribute supplies the value for the “synchronizing DLCEP” parameter of the DLL.

9.1.4.2.21 ReceiverTimeliness

9.1.4.2.22 SubscriberDlTimelinessClass

This optional attribute provides the timeliness class of a Subscriber provided by the DLL. This attribute supplies the value for the “DL-timeliness-class” parameter of the DLL. The permitted values Residence, Update, Synchronous, and None correspond to RESIDENCE, UPDATE, SYNCHRONOUS, and NONE defined by the DLL.

9.1.4.2.23 SubscriberTimeWindowSize

This optional attribute provides time window of a Subscriber provided by the DLL. This attribute supplies the value for the “Time window size” parameter of the DLL.

9.1.4.2.24 SubscriberSynchronizingDlcep

This optional attribute is present when the SubscriberDlTimelinessClass attribute has the value of Update or Synchronous and provides a DLCEP that is used to generate synchronizing events by the DLL. The FAS user may derive the AREP from which the synchronizing events are delivered by this attribute value. This attribute supplies the value for the “synchronizing DLCEP” parameter of the DLL.

9.1.4.2.25 LasScheduled

This attribute specifies whether this AREP is LAS-scheduled or not. The value of True means that this APRE is LAS-Scheduled. The value of False means that it is not LAS-Scheduled.

NOTE This attribute does not affect the operation of the FAS and the DLL. It is used by the FAS users only..

9.1.4.3 DLL services

Refer to to IEC 61158-3-1 and IEC 61158-4-1 for DLL service descriptions.

9.2 Application relationship protocol machines (ARPMs)

9.2.1 AR protocol machine (ARPM) for QUU AREP

9.2.1.1 QUU ARPM states

The defined states and their descriptions of the QUU ARPM are shown in Table 12 and Figure 24.

Table 12 – QUU ARPM states

CLOSED	The AREP is defined, but not capable of sending or receiving FAS-PDUs. It may send or receive Associate service FAS-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAS-PDUs.

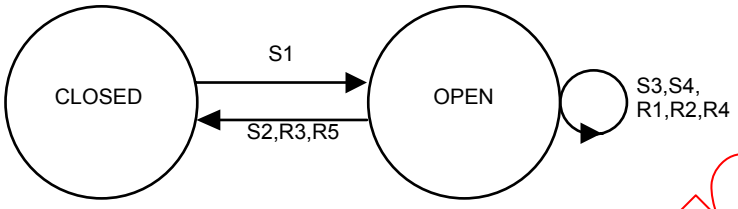


Figure 24 – State transition diagram of the QUU ARPM

9.2.1.2 QUU ARPM state table

Table 13 and Table 14 specify the QUU QRPM state machine.

Table 13 – QUU ARPM state table – sender transactions

#	Current state	Event or condition => action	Next state

S1	CLOSED	ASC_req
		=>
		ASC_cnf(+) {
		arep_id := GetArepId (),
		user_data := "null"
		}
			OPEN

S2	OPEN	Abort_req => (no actions taken)	CLOSED

S3	OPEN	DTU_req
		&& ConfigurationType = "Linked"
		&& Role = "Source"
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Unitdata_req",
		arep_id := GetArepId (),
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "DTU_PDU",
		fas_data := user_data)
		}
			OPEN

S4	OPEN	DTU_req
		&& ConfigurationType = "Free"
		&& Role = "Source"
		=>
		RemoteDlsapAddress := remote_dlsap_address,
		
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Unitdata_req",
		arep_id := GetArepId (),
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "DTU_PDU",
		fas_data := user_data)
		}
			OPEN

Table 14 – QUU ARPM state table – receiver transactions

#	Current state	Event or condition => action	Next state

R1	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Unitdata_ind”
		&& ConfigurationType = “Linked”
		&& RemoteDlsapAddress = calling_address
		&& Role = “Sink”
		&& FAS_Pdu_Type (fas_pdu) = “DTU_PDU”
		=>
		DTU_ind {
		arep_id := GetArepId (),
		user_data := fas_pdu
		}
			OPEN

R2	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Unitdata_ind”
		&& ConfigurationType = “Free”
		&& Role = “Sink”
		&& FAS_Pdu_Type (fas_pdu) = “DTU_PDU”
		=>
		DTU_ind {
		arep_id := GetArepId (),
		remote_dlsap_address := calling_address,
		user_data := fas_pdu
		}
			OPEN

R3	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Unitdata_ind”
		&& Role = “Sink”
		&& FAS_Pdu_Type (fas_pdu) <> “DTU_PDU”
		=>
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := "FAS",
		reason_code := " Invalid FAS-PDU",
		additional_detail := "null"
		}
			CLOSED

R4	OPEN	ErrorToARPM => (no actions taken)	OPEN

R5	OPEN	FAS-PDU_ind
		&& Role = “Source”
		=>
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid Event for Role”,
		additional_detail := "null"
		}
			CLOSED

9.2.2 AR protocol machine (ARPM) for QUB AREP

9.2.2.1 QUB ARPM states

The defined states and their descriptions of the QUB ARPM are shown in Table 15 and Figure 25.

Table 15 – QUB ARPM states

CLOSED	The AREP is defined, but not capable of sending or receiving FAS-PDUs. It may send or receive Associate service FAS-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAS-PDUs.
REQUESTING (REQ)	The AREP has sent an Associate Request FAS-PDU and is waiting for a response from the remote AREP.
RESPONDING (RSP)	The AREP has received an Associate Request FAS-PDU, delivered an Associate.indication primitive and is waiting for a response from its user.
REPLIED (REPL)	The Server AREP has issued an ASC_rsp(+) primitive and is waiting for receiving a "connection-established" indication from the DLL.
SAME	Indicates that the next state is the same as the current state.

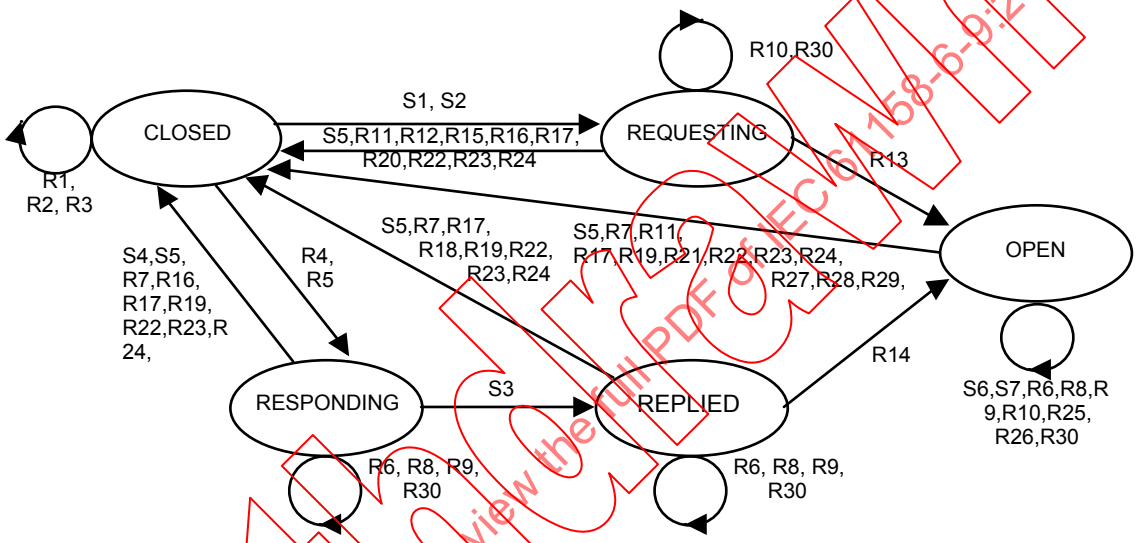


Figure 25 – State transition diagram of QUB ARPM

9.2.2.2 QUB ARPM state table

Table 16 and Table 17 specify the QUB ARPM state machine.

Table 16 – QUB ARPM state table – sender transactions

#	Current state	Event or condition => action	Next state

S1	CLOSED	ASC_req
		&& Initiator = “True”
		&& RemoteAddressConfigurationType := “Free”
		RemoteDlcepAddress := remote_dlcep_address,
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Connect_req",
		arep_id := GetArepId (),
		called_address := “default dlsap address”,
		calling_address := “default dlsap address”,
		local_dlcep_address := LocalDlcep,
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "ASC_ReqPDU",
		calling_dlcep_address := LocalDlcepAddress,
		called_dlcep_address := RemoteDlcepAddress,
		fas_data := user_data)
		}
			REQ

S2	CLOSED	ASC_req
		&& Initiator = “True”
		&& RemoteAddressConfigurationType := “Linked”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Connect_req",
		arep_id := GetArepId (),
		called_address := “default dlsap address”,
		calling_address := “default dlsap address”,
		local_dlcep_address := LocalDlcep,
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "ASC_ReqPDU",
		calling_dlcep_address := LocalDlcepAddress,
		called_dlcep_address := RemoteDlcepAddress,
		fas_data := user_data)
		}
			REQ

S3	RSP	ASC_rsp(+)
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Connect_rsp",
		arep_id := GetArepId (),
		responding_address := “default dlsap address”,
		local_dlcep_address := LocalDlcep,
		dlcep_dl_id := DlcepDlIdentifier,
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "ASC_RspPDU",
		fas_data := user_data)
		}
			REPL

S4	RSP	ASC_rsp(-)
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Disconnect_req",
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "connection rejection–transient condition",
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "ASC_ErrPDU",
		fas_data := user_data)
		}
			CLOSED

S5	NOT CLOSED	Abort_req
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Disconnect_req",
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "disconnection–normal condition",
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "Abort_PDU",
		fas_id := identifer,
		fas_reason_code := reason_code,
		fas_additional_detail := additional_detail)
		}
			CLOSED

#	Current state	Event or condition => action	Next state

S6	OPEN	DTC_req
		&& Role = “Client” || “Peer”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Data_req",
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "DTC_ReqPDU",
		fas_data := user_data)
		}
			OPEN

S7	OPEN	DTC_rsp
		&& Role = “Server” || “Peer”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Data_req",
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "DTC_RspPDU",
		fas_data := user_data)
		}
			OPEN

Table 17 – QUB ARPM state table – receiver transactions

#	Current state	Event or condition => action	Next state

R1	CLOSED	Connect_ind
		&& Initiator = “True”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Multiple Initiators",
		dlsdu := “null”
		}
			CLOSED

R2	CLOSED	Connect_ind
		&& Initiator = “False”
		&& FAS_Pdu_Type (dls_user_data) <> “ASC_ReqPDU”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := “Invalid FAS-PDU”,
		dlsdu := “null”
		}
			CLOSED

R3	CLOSED	Connect_ind
		&& Initiator = “False”
		&& FAS_Pdu_Type (dls_user_data) = “ASC_ReqPDU”
		&& RemoteAddressConfigurationType = “Linked”
		&& RemoteDlcepAddress <> calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Remote Address Mismatch",
		dlsdu := “null”
		}
			CLOSED

#	Current state	Event or condition => action	Next state

R4	CLOSED	Connect_ind
		&& Initiator = “False”
		&& FAS_Pdu_Type (dls_user_data) = “ASC_ReqPDU”
		&& RemoteAddressConfigurationType = “Free”
		=>
		RemoteDlcepAddress := calling_dlcep_address,
		DlcepDlIdentifier := dlcep_dl_id,
		MaxDlsduSizeFromRequesterNegotiated := dlsdu_size_from_requester,
		MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder,
		
		ASC_ind {
		arep_id := GetArepId (),
		user_data := dls_user_data
		}
			RSP

R5	CLOSED	Connect_ind
		&& Initiator = “False”
		&& FAS_Pdu_Type (dls_user_data) = “ASC_ReqPDU”
		&& RemoteAddressConfigurationType = “Linked”
		&& RemoteDlcepAddress = calling_dlcep_address
		=>
		DlcepDlIdentifier := dlcep_dl_id,
		MaxDlsduSizeFromRequesterNegotiated := dlsdu_size_from_requester,
		MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder,
		
		ASC_ind {
		arep_id := GetArepId (),
		user_data := dls_user_data
		}
			RSP

R6	RSP
	REPL
	OPEN
		Connect_ind
		&& Initiator = “False”
		&& RemoteAddressConfigurationType = “Linked”
		&& RemoteDlcepAddress <> calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Remote Address Mismatch",
		dlsdu := “null”
		}
			SAME

R7	RSP
	REPL
	OPEN
		Connect_ind
		&& Initiator = “False”
		&& RemoteDlcepAddress = calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Invalid FAS-PDU",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := "FAS",
		reason_code := "Invalid FAS-PDU"
		}
			CLOSED

R8	RSP
	REPL
	OPEN
		Connect_ind
		&& Initiator = “False”
		&& FAS_Pdu_Type (dls_user_data) = “ASC_ReqPDU”
		&& RemoteAddressConfigurationType = “Free”
		&& RemoteDlcepAddress <> calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "AREP Busy",
		dlsdu := “null”
		}
			SAME

#	Current state	Event or condition => action	Next state

R9	RSP
	REPL
	OPEN
		Connect_ind
		&& Initiator = “False”
		&& FAS_Pdu_Type (dls_user_data) <> “ASC_ReqPDU”
		&& RemoteAddressConfigurationType = “Free”
		&& RemoteDlcepAddress <> calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Invalid FAS-PDU",
		dlsdu := “null”
		}
			SAME

R10	REQ
	OPEN
		Connect_ind
		&& Initiator = “True”
		&& RemoteDlcepAddress <> calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Multiple Initiators",
		dlsdu := “null”
		}
			SAME

R11	REQ
	OPEN
		Connect_ind
		&& Initiator = “True”
		&& RemoteDlcepAddress = calling_dlcep_address
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := dlcep_dl_id (from Connect_ind),
		reason := "Multiple Initiators",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := "FAS",
		reason_code := "Multipe Initiators"
		}
			CLOSED

R12	REQ	Connect_cnf
		&& FAS_Pdu_Type (dls_user_data) <> “ASC_RspPDU”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid FAS-PDU",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := "FAS",
		reason_code := "Invalid FAS-PDU"
		}
			CLOSED

R13	REQ	Connect_cnf
		&& FAS_Pdu_Type (dls_user_data) = “ASC_RspPDU”
		=>
		MaxDlsduSizeFromRequesterNegotiated := dlsdu_size_from_requester,
		MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder,
		DllPriorityNegotiated := dll_priority,
		
		ASC_cnf(+) {
		arep_id := GetArepId (),
		user_data := dls_user_data
		}
			OPEN

R14	REPL	FAS-PDU_ind && dmpm_service_name = “DMPM_Connection_Established_ind” => (no actions taken)	OPEN

#	Current state	Event or condition => action	Next state

R15	REQ	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& FAS_Pdu_Type (fas_pdu) = “ASC_ErrPDU”
		=>
		ASC_cnf(-) {
		arep_id := GetArepId (),
		user_data := dls_user_data
		}
			CLOSED

R16	REQ
	RSP
		FAS-PDU_ind
		&& dmpm_service_name <> “DMPM_Disconnect_ind”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid DL-Event",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := "FAS",
		reason_code := "Invalid DL-Event"
		}
			CLOSED

R17	NOT CLOSED	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& fas_pdu <> “null”
		&& FAS_Pdu_Type (fas_pdu) = “Abort_PDU”
		=>
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "False",
		identifier := AbortIdentifier (fas_pdu),
		reason_code := AbortReason (fas_pdu),
		additional_detail := AbortDetail (fas_pdu)
		}
			CLOSED

R18	REPL	FAS-PDU_ind
		&& ((dmpm_service_name <> “DMPM_Disconnect_ind”)
		&& (dmpm_service_name <>“DM_Connection_Established_ind”))
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid DL-Event",
		dlsdu := “null”
		},
		
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid DL-Event”,
		additional_detail := “null”
		}
			CLOSED

R19	REPL
	RSP
	OPEN
		FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& fas_pdu <> “null”
		&& FAS_Pdu_Type (fas_pdu) <> “Abort_PDU”
		=>
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid FAS-PDU”,
		additional_detail := “null”
		}
			CLOSED

#	Current state	**Event** or **condition** **=> action**	Next state

R20	REQ	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& fas_pdu <> “null”
		&& ((FAS_Pdu_Type (fas_pdu) <> “Abort_PDU”)
		&& (FAS_Pdu_Type (fas_pdu) <> “ASC_ErrPDU”))
		=>
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid FAS-PDU”,
		additional_detail := “null”
		}
			CLOSED

R21	OPEN	FAS-PDU_ind
		&& ((dmpm_service_name <> “DMPM_Disconnect_ind”)
		&& (dmpm_service_name <> “DMPM_Data_ind”))
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid DL-Event",
		dlsdu := “null”
		},
		
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid DL-Event”,
		additional_detail := “null”
		}
			CLOSED

R22	NOT CLOSED	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& fas_pdu = "null"
		&& originator = "remote_dls_provider"
		=>
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "False",
		identifier := “Data-link layer”,
		reason_code := reason,
		additional_detail := "null"
		}
			CLOSED

R23	NOT CLOSED	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& fas_pdu = "null"
		&& originator = "remote_dls_user"
		=>
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "False",
		identifier := “FAS”,
		reason_code := reason,
		additional_detail := "null"
		}
			CLOSED

R24	NOT CLOSED	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Disconnect_ind”
		&& fas_pdu = "null"
		&& originator = "local_dls_provider"
		=>
		Abort_ind{
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “Data-link layer”,
		reason_code := reason,
		additional_detail := "null"
		}
			CLOSED

#	Current state	Event or condition => action	Next state

R25	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Data_ind”
		&& Role = “Peer” || “Server”
		&& FAS_Pdu_Type (fas_pdu) = “DTC_ReqPDU”
		=>
		DTC_ind {
		arep_id := GetArepId (),
		user_data := fas_pdu
		}
			OPEN

R26	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Data_ind”
		&& Role = “Client” || “Peer”
		&& FAS_Pdu_Type (fas_pdu) = “DTC_RspPDU”
		=>
		DTC_cnf {
		arep_id := GetArepId (),
		user_data := fas_pdu
		}
			OPEN

R27	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Data_ind”
		&& Role = “Server”
		&& FAS_Pdu_Type (fas_pdu) <> “DTC_ReqPDU”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid FAS-PDU",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := "Invalid FAS-PDU",
		additional_detail := "null"
		}
			CLOSED

R28	OPEN	FAS-PDU_ind
		&& dmpm_service_name = “DMPM_Data_ind”
		&& Role = “Client”
		&& FAS_Pdu_Type (fas_pdu) <> “DTC_RspPDU”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid FAS-PDU",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid FAS-PDU”,
		additional_detail := "null"
		}
			CLOSED

#	Current state	Event or condition => action	Next state

R29	OPEN	FAS-PDU_ind
		&& Role = “Peer”
		&& dmpm_service_name = “DMPM_Data_ind”
		&& ((FAS_Pdu_Type (fas_pdu) <> “DTC_ReqPDU”)
		&& (FAS_Pdu_Type (fas_pdu) <> “DTC_RspPDU”))
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Disconnect_req”,
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "Invalid FAS-PDU",
		dlsdu := “null”
		},
		
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid FAS-PDU”,
		additional_detail := "null"
		}
			CLOSED

R30	NOT CLOSED	ErrorToARPM => (no actions taken)	SAME
NOTE It is a local matter to report an error status to network management entities. The ARPM does not abort the existing connections. The FAS user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive.			

9.2.3 AR protocol machine (ARPM) for BNU AREP

9.2.3.1 BNU ARPM states

The defined states and their descriptions of the BNU ARPM are shown in Table 18 and Figure 26.

Table 18 – BNU ARPM states

CLOSED	The AREP is defined, but not capable of sending or receiving FAS-PDUs.
REQUESTING	The AREP has issued an ASC_req and waiting for an ASC_cnf primitive.
OPEN	The AREP is defined and capable of sending or receiving FAS-PDUs.

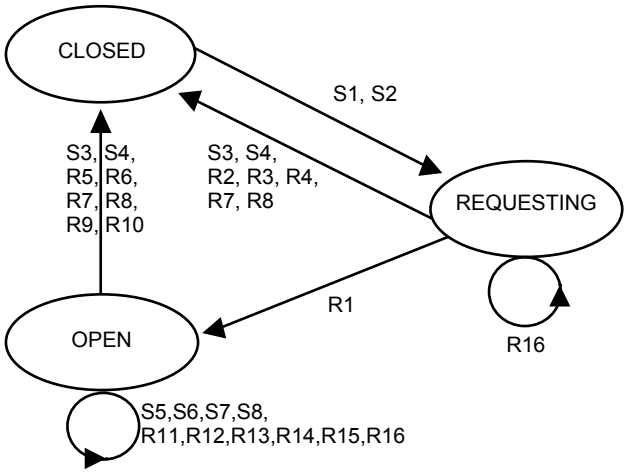


Figure 26 – State transition diagram of the BNU ARPM

9.2.3.2 BNU ARPM state table

Table 19 and Table 20 specify the BNU ARPM state machine.

Table 19 – BNU ARPM state table – sender transactions

#	Current state	Event or condition => action	Next state

S1	CLOSED	ASC_req
		&& Role = “Publisher”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Connect_req",
		arep_id := GetArepId (),
		called_address := “null”,
		calling_address := “default dlsap address”,
		local_dlcep_address := PublisherDlcepAddress,
		dlsdu := "null"
		}
			REQ

S2	CLOSED	ASC_req
		&& Role = “Subscriber”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Connect_req",
		arep_id := GetArepId (),
		called_address := PublisherDlcepAddress,
		calling_address := “default dlsap address”,
		local_dlcep_address := “default subscriber dlcep address”,
		dlsdu := "null"
		}
			REQ

S3	NOT CLOSED	Abort_req
		&& Role = “Publisher”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Disconnect_req",
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "disconnection–normal condition",
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := “Abort_PDU”,
		fas_id := identifer,
		fas_reason_code := reason_code,
		fas_additional_detail := additional_detail)
		}
			CLOSED

S4	NOT CLOSED	Abort_req
		&& Role = “Subscriber”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Disconnect_req",
		arep_id := GetArepId (),
		dlcep_dl_id := DlcepDlIdentifier,
		reason := "disconnection–normal condition",
		dlsdu := “null”
		}
			CLOSED

S5	OPEN	DTU_req
		&& Role = “Publisher”
		=>
		FAS-PDU_req {
		dmpm_service_name := "DMPM_Put_req",
		arep_id := GetArepId (),
		dlsdu := BuildFAS-PDU (
		fas_pdu_name := "DTU_PDU",
		fas_sdu := user_data)
		}
			OPEN

S6	OPEN	FCMP_req
		&& Role = “Publisher”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Compel_Service_req”,
		arep_id := GetArepId (),
		action_class := “Local”
		}
			OPEN

#	Current state	Event or condition => action	Next state

S7	OPEN	FCMP_req
		&& Role = “Subscriber”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Compel_Service_req”,
		arep_id := GetArepId (),
		action_class := “Remote”
		}
			OPEN

S8	OPEN	GBM_req
		&& Role = “Subscriber”
		=>
		FAS-PDU_req {
		dmpm_service_name := “DMPM_Get_req”,
		arep_id := GetArepId ()
		}
			OPEN

Table 20 – BNU ARPM state table – receiver transactions

#	Current state	Event or condition => action	Next state

R1	REQ	Connect_cnf
		=>
		ASC_cnf(+) {
		arep_id := GetArepId (),
		user_data := "null"
		}
			OPEN

R2	REQ	FAS-PDU_ind
		&& Role = “Subscriber”
		&& dmpm_service_name = "DMPM_Disconnect_ind"
		&& fas_pdu = “null”
		&& originator = "local_dls_provider"
		=>
		ASC_cnf(-) {
		arep_id := GetArepId (),
		user_data := "null"
		}
			CLOSED

R3	REQ	FAS-PDU_ind
		&& Role = “Subscriber”
		&& dmpm_service_name = "DMPM_Disconnect_ind"
		&& fas_pdu = "null"
		&& originator = "remote_dls_provider"
		=>
		ASC_cnf(-) {
		arep_id := GetArepId (),
		user_data := "null"
		}
			CLOSED

R4	REQ	FAS-PDU_ind
		&& Role = “Subscriber”
		&& dmpm_service_name <> "DMPM_Disconnect_ind"
		=>
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “FAS”,
		reason_code := “Invalid DI Event”,
		additional_detail := "null"
		}
			CLOSED

R5	OPEN	FAS-PDU_ind
		&& Role = “Subscriber”
		&& dmpm_service_name = "DMPM_Disconnect_ind"
		&& fas_pdu = “null”
		&& originator = "local_dls_provider"
		=>
		Abort_ind {
		arep_id := GetArepId (),
		locally_generated := "True",
		identifier := “Data-link layer”,
		reason_code := reason,
		additional_detail := "null"
		}
			CLOSED