PUBLICLY
AVAILABLE
SPECIFICATION

IEC PAS 62405

First edition
2005-06

Real-time Ethernet Vnet/IP™ specification

Reference number
IEC/PAS 62405:2005(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/ justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

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

PUBLICLY
AVAILABLE
SPECIFICATION

IEC PAS 62405

First edition
2005-06

Real-time Ethernet Vnet/IP™ specification

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XG

For price, see current catalogue

CONTENTS

Section 3: Application Layer protocol specification

Part 4: Data Link Service definition

INTERNATIONAL ELECTROTECHNICAL COMMISSION

Real-time Ethernet Vnet/IP™ specification

FOREWORD

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

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this document may involve the use of a patent concerning Vnet/IP[1].

The Vnet/IP has the patent applications listed below:

PCT Application No. PCT/JP2004/011537

PCT Application No. PCT/JP2004/011538.

IEC takes no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured the IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with IEC. Information may be obtained from:

Yokogawa Electric Corporation
Intellectual Property & Standardization Center
2-9-32 Nakacho, Musashino-shi, 180-8750
Tokyo, Japan.

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

---

[1] Vnet/IP is the trade name of Yokogawa Electric Corporation. This information is given for the convenience of users of this PAS and does not constitute an endorsement by IEC of the trademark holder or any of its products. Compliance to this profile does not require use of the trade name Vnet/IP. Use of the trade name Vnet/IP requires permission of the Yokogawa Electric Corporation.

A PAS is a technical specification not fulfilling the requirements for a standard but made available to the public .

IEC-PAS 62405 has been processed by subcommittee 65C: Digital communications, of IEC technical committee 65: Industrial-process measurement and control.

The text of this PAS is based on the following document:	This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document
Draft PAS	**Report on voting**
65C/352/NP	65C/369/RVN

Following publication of this PAS, the technical committee or subcommittee concerned will transform it into an International Standard.

It is intended that the content of this PAS will be incorporated in the future new edition of the various parts of IEC 61158 series according to the structure of this series.

This PAS shall remain valid for an initial maximum period of three years starting from 2005-06. The validity may be extended for a single three-year period, following which it shall be revised to become another type of normative document or shall be withdrawn.

Real-time Ethernet Vnet/IP™ specification

Section 1: Overview

This PAS has been divided into five sections.

Section 1: Overview

Section 2: Application Layer Service definition

Section 3: Application Layer protocol specification

Section 4: Data Link Layer Service definition

Section 5: Data Link Layer protocol specification

1 Introduction

The Vnet/IP is designed for highly reliable real-time communication and applies to the operation of process control systems. The process control system consists of many controllers, human-machine interfaces, process monitors, and gateways to external networks which are connected to Ethernet-based IP networks. The data transfer between these devices is performed by peer-to-peer or multicast communication.

It is required that the process control system responds within a deterministic time. At the same time, it is also required that multi-vendor's multi-devices can be connected to the same IP network. For example, the process control system shall handle operational signals for the actuator within a determined time and needs to communicate with other devices connected to Ethernet-based IP networks to exchange non-realtime information such as files and images.

The Vnet/IP realizes these two conflicting requirements on the same network using a unique technology as described in this PAS.

2 Scope and objective

This PAS describes the communication profile of the whole Vnet/IP, services of each layer and the specifications of the protocols for each layer.

3 Structure of this document

Section 1 of this PAS presents an overview of and guidance for the Vnet/IP Specification. It also explains the structure and content of the Vnet/IP Specification and shows how to use each section of the document. In addition, it specifies the communication profile of the Vnet/IP.

Sections 2 and 3 present the Vnet/IP Specification for the Application Layer, and Sections 4 and 5 for the Data Link Layer. This PAS refers the appropriate international standards for the specifications of other layers.

The Data Link and Application Layers are described in complementary ways, in terms of the services offered and the protocol which provides those services.

Table 1 shows the differences between service and protocol viewpoints of the Data Link and Application Layers. The protocol parts show the layer implementer's view and the service parts show the layer user's view.

Table 1 – Concept of DL/AL to separate service and protocol parts

Layer user Oriented view	**Layer implementer** Oriented view
AL Services (Section 2 of this PAS) - Model and concepts - Data type definitions - Application objects - Service description - Communication endpoint management.	**AL Protocol (Section 3 of this PAS)** - Syntax definition and coding - Application relationships procedures - Protocol machines (state machines)
DL Services (Section 4 of this PAS) - Model and concepts - Service description - Management services	**DL Protocol (Section 5 of this PAS)** - Coding - Protocol machines (state machines)

The Application Layer structure is as follows:

- "What" is described by Application Service Elements (ASE); and
- "How" is described by Application Relationships (AR).

The Data Link Layer structure is as follows:

- "What" is described by Data Link services and models; and
- "How" is described by Data Link protocol machines and medium access principles.

4 Normative reference

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

IEC 61158-1:2003, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 1: Overview and guidance for the IEC 61158 series*

IEC 61158-3:2003, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 3: Data link service definition*

IEC 61158-4:2003, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 4: Data link protocol specification*

IEC 61158-5:2003, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 5: Application layer service definition*

IEC 61158-6:2003, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 6: Application layer protocol specification*

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

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

ISO/IEC 9545:1994, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 8824-2:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*

ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding RUles (CER) and Distinguished Encoding RUles (DER)*

ISO/IEC 8802-0:2000, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks - Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

Internet Engineering Task Force (IETF), *Request for Comments (RFC):*

RFC 768 *User Datagram Protocol*
(available at <http://www.ietf.org/rfc/rfc0768.txt>)
RFC 791 *Internet Protocol*
(available at <http://www.ietf.org/rfc/rfc0791.txt>)
RFC 792 *Internet Control Message Protocol*
(available at <http://www.ietf.org/rfc/rfc0792.txt>)
RFC 826 *Ethernet Address Resolution Protocol*
(available at <http://www.ietf.org/rfc/rfc0826.txt>)
RFC 894 A standard for the Transmission of IP Datagrams over Ethernet Networks
(available at <http://www.ietf.org/rfc/rfc0894.txt>)
RFC 1112 *Host Extensions for IP Multicasting*
(available at <http://www.ietf.org/rfc/rfc1112.txt>)
RFC 2236 *Internet Group Management Protocol Version 2*
(available at <http://www.ietf.org/rfc/rfc2236.txt>)

5 Definitions

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

5.1 Terms and definitions

5.1.1 ISO/IEC 7498-1 terms

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

1) application entity
2) application protocol data unit
3) application service element

5.1.2 ISO/IEC 8824-2 terms

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

1) any type
2) bitstring type
3) Boolean type
4) choice type
5) false
6) integer type
7) null type
8) octetstring type
9) sequence of type
10) sequence type
11) simple type
12) structured type
13) tagged type
14) true
15) type
16) value

5.1.3 ISO/IEC 10731 terms

1) (N)-connection
2) (N)-entity
3) (N)-layer
4) (N)-service
5) (N)-service-access-point
6) application-service-object
7) confirm (primitive)
8) deliver (primitive)
9) indication (primitive)
10) OSI-confirmed-facility
11) OSI-facility
12) OSI-local view
13) OSI-mandatory-facility
14) OSI-non-confirmed-facility
15) OSI-provider-initiated-facility
16) OSI-provider-optional-facility
17) OSI-service
18) OSI-service primitive; primitive
19) OSI-service-provider
20) OSI-service-user
21) OSI-user-optional-facility
22) request (primitive)
23) response (primitive)
24) submit (primitive)

5.1.4 IEC 61158-3 and IEC 61158-4 terms

For the purposes of this document, the following terms as defined in IEC 61158-3 and IEC 61158-4 apply.

1) DL-Time
2) DL-Scheduling-policy
3) DL-connection-oriented mode
4) fixed tag
5) generic tag
6) link
7) MAC ID
8) network address
9) node address
10) node
11) tag
12) scheduled
13) unscheduled

5.1.5 IEC 61158-5 terms

For the purposes of this document, the following terms as defined in IEC 61158-5 apply.

1) active connection control object
2) application
3) application objects
4) application process
5) application process identifier
6) application relationship
7) application relationship application service element
8) application relationship endpoint
9) attribute
10) behaviour
11) channel
12) class
13) client
14) connection
15) connection point
16) conveyance path
17) cyclic
18) dedicated AR
19) device
20) end node
21) endpoint
22) error
23) error class
24) event
25) group
26) instance
27) instantiated
28) interface
29) invocation
30) management information
31) method
32) network
33) object
34) originator
35) peer
36) produce
37) producer
38) provider
39) publisher
40) resource

41) server
42) service
43) slot
44) subscriber
45) sync

5.1.6 IEC 61158-6 terms

For the purposes of this document, the following terms as defined in IEC 61158-6 apply.

1) called
2) calling
3) receiving
4) sending

5.1.7 Other terms and definitions

5.1.7.1 Vnet/IP Network

a network that is compliant with the Vnet/IP specification, which consists of one or more domains. The domains are connected to each other by routers. In a redundant Vnet/IP network structure, the Vnet/IP network consists of two networks: a primary network and a secondary network

NOTE The primary network and the secondary network are independent and are not connected to each other.

5.1.7.2 domain

a part of the Vnet/IP network consisting of one or two subnetwork(s)

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

5.1.7.3 subnetwork

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

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

5.1.7.4 router

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

5.1.7.5 route

a logical communication channel between two communication end nodes

5.1.7.6 segment

a communication channel that connects two nodes directly without intervening bridges

5.1.7.7 bridge

intermediate equipment that connects two or more segments using a Data Link layer relay function

5.1.7.8 internal bridge

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

5.1.7.9 junction bridge

a bridge to which at least one router, external bridge or node non-compliant with this specification, and to which at least one internal bridge or Vnet/IP station is connected

5.1.7.10 external bridge

a bridge to which neither internal bridges nor Vnet/IP stations are connected directly

5.1.7.11 station

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

5.1.7.12 Vnet/IP station

a station compliant with this specification

5.1.7.13 redundant station

a station that consists of a pair of end nodes

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

5.1.7.14 non-redundant station

a station that consists of a single end node

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

5.1.7.15 link

a physical communication channel between two nodes

5.1.7.16 path

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

5.1.7.17 interface port

a physical connection point of an end node, which has an independent DL-address

5.1.7.18 redundant interface node

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

5.1.7.19 non-redundant interface node

a node whch has a single interface port

5.1.7.20 domain master

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

5.1.7.21 network time master

a station which distributes network time to domain masters

5.1.7.22 domain number

a numeric identifier which indicates a domain

5.1.7.23 station number

a numeric identifier which indicates a Vnet/IP station

5.2 Abbreviations and symbols

5.2.1 ISO/IEC 10731 abbreviations

ASE application-service-element
ASO application-service-object
OSI Open Systems Interconnection

5.2.2 IEC 61158-3 abbreviations and symbols

DL- Data Link layer (as a prefix)
DLE DL-entity (the local active instance of the Data Link layer)
DLL DL-layer
DLM DL-management
DLMS DL-management Service
DLPDU DL-protocol-data-unit
DLS DL-service
DLSAP DL-service-access-point
DLSDU DL-service-data-unit
FIFO First-in first-out (queuing method)
QoS Quality of service

5.2.3 IEC 61158-4 abbreviations and symbols

LLC logical link control
MAC medium access control
Ph- physical layer (as a prefix)
PhL Ph-layer

5.2.4 IEC 61158-5 abbreviations and symbols

ACCO Active Connection Control Object
AE Application Entity
AL Application Layer
ALP Application Layer Protocol
AP Application Process
Ap_ Prefix for Data types defined for AP ASE
APDU Application Protocol Data Unit
API Application Process Identifier
APO Application Object
AR Application Relationship

AREP	Application Relationship End Point
ASN.1	Abstract Syntax Notation One
BCD	Binary Coded Decimal
BER	Basic Encoding Rule
CID	Connection ID
CIM	Computer Integrated Manufacturing
Cnf	Confirmation
cnf	confirmation primitive
Dl_	Prefix for Data types defined for Data Link Layer types DL Data Link
Dt_	Prefix for Data types defined for Data Type ASE
Er_	Prefix for Error types defined
Err	Error (used to indicate an APDU type)
Ev_	Prefix for Data types defined for Event ASE
FAL	Fieldbus Application Layer
Gn_	Prefix for Data Types defined for general use
ID	Identifier
IEC	International Electrotechnical Commission
Ind	Indication
ind	indication primitive
IP	Internet Protocol
ISO	International Organization for Standardization
lsb	least significant bit
msb	most significant bit
out	output primitive
PDU	Protocol Data Unit
PL	Physical Layer
Req	Request
req	request primitive
Rsp	Response
rsp	response primitive
SAP	Service Access Point
SDU	Service Data Unit
ToS	Type Of Service

5.2.5 Vnet/IP abbreviations and symbols

ARPM	Application Relationship Protocol Machine
ASS	Acknowledged Sequence of unitdata transfer Service
AUS	Acknowledged Unitdata transfer Service
FSPM	FAL Service Protocol Machine
MSS	Multipoint Sequence of unitdata transfer Service
MSU-AR	Multipoint network-Scheduled Unconfirmed publisher/subscriber AREP
MTU-AR	Multipoint user-Triggered Unconfirmed publisher/subscriber AREP
MUS	Multipoint Unitdata transfer Service

PSU-AR Point-to-point network-Scheduled Unconfirmed client/server AREP

PTC-AR Point-to-point user-Triggered Confirmed client/server AREP

PTU-AR Point-to-point user-Triggered Unconfirmed client/server AREP

UUS Unacknowledged Unitdata transfer Service

5.3 Conventions

5.3.1 General conventions

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

5.3.2 Conventions for communication profile definitions

This PAS uses the descriptive conventions given in IEC 61784 for communication profile definitions.

5.3.3 Conventions for class definitions

This PAS uses the descriptive conventions given in IEC 61158-5 for class definitions.

5.3.4 Conventions for FAL service definitions

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

5.3.5 Conventions for APDU abstract syntax definitions

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

5.3.6 Conventions for APDU transfer syntax definitions

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

5.3.7 Conventions for AE state machine definitions

The conventions used for AE state machine definitions are described in Table 2.

Table 2 – Conventions used for AE state machine definitions

#	Current state	Event / condition => action	Next state
Name of this transition.	The current state to which this state transition applies.	Events or conditions that trigger this state transition. => The actions that are taken when the above events or conditions are met. The actions are always indented below events or conditions.	The next state after the actions in this transition are taken.

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

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

xxx Parameter name.

Example:

Identifier := reason

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

“xxx” Indicates fixed value.

Example:

Identifier := “abc”

means value “abc” is assigned to a parameter named ‘Identifier.’

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

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

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

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

&& Logical “AND”

|| Logical “OR”

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

for

endfor

if

else

elseif

The following shows examples of description using the reserved words.

Example 1:

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

Example 2:

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

5.3.8 Conventions for DLL service definitions

This standard uses the descriptive conventions given in IEC 61158-3 for DLL service definitions.

5.3.9 Conventions for DLE protocol procedure definitions

The conventions used for DLE state machine definitions are described in Table 3.

Table 3 – Conventions used for protocol procedure definitions

Event	Condition	Procedure
Events that trigger these actions	Conditions	Actions that are taken when the events and conditions are met.

6 Communication Profile of Vnet/IP

6.1 General overview

The Vnet/IP protocol is described using the principles, methodology and model of ISO/IEC 7498. In addition, it also follows the three-layer basic fieldbus reference model described in the IEC 61158-1.

The OSI model provides a layered approach to communication standards, whereby the layers can be developed and modified independently. The Vnet/IP communication profile is based on the three-layer structure, and each layer of OSI seven layers is mapped onto these three layers as follows.

Functions of the intermediate OSI layers, layers 5 – 6, are consolidated into the Application Layer.

Functions of the intermediate OSI layers, layers 3 – 4, are consolidated into the Data Link Layer.

Likewise, some features common to users of the Fieldbus Application Layer are provided to simplify user operation.

Table 4 shows the OSI layers, their functions and the equivalent layers in the Vnet/IP layer model.

Table 4 – OSI layers and Vnet/IP layers

OSI layer	Function	Vnet/IP layer
7 Application	Translates demands placed on the communication stack into a form understood by the lower layers and vice versa	Application
6 Presentation	Converts data to/from standardized network formats	
5 Session	Synchronizes and manages data	
4 Transport	Provides transparent reliable data transfer	Data link
3 Network	Performs message routing	
2 Data link	Controls access to the communication medium. Performs error detection	
1 Physical	Encodes/decodes signals for transmission/reception in a form appropriate to the communication medium. Specifies communication media characteristics.	Physical

Table 5 shows an overview of the communication profile for Vnet/IP.

Table 5 – Overview of Vnet/IP profile

Layer	Protocol
Application	The specification of this PAS
Transport	RFC 768 and the specifications in this PAS
Network	RFC 791
Data Link	ISO/IEC 8802-3, -2, -10
Physical	any of ISO/IEC 8802-3

6.2 Physical Layer

ISO/IEC 8802-3 shall be used.

6.3 Data Link Layer

6.3.1 MAC sublayer

ISO/IEC 8802-3 shall be used.

6.3.2 LLC sublayer

ISO/IEC 8802-3 shall be used.

6.4 Network Layer

Internet standard RFC 791 (IP, Internet Protocol) and its amendments and successors shall be used.

6.4.1 IP Address

Unicast address
IPv4 class C private address scope shall be used. Each subnetwork of a domain has the respective IP network address as follows.

IP address for the interface connected to the primary network :

192.168.(192+Domain number).(Host address)

IP address for the interface connected to the secondary network :

192.168.(224+Domain number).(Host address)

Each node of a redundant station has a respective IP host address as follows.

Host address: Station number or Station number + 64

Group address
IPv4 class D Organization-Local Scope shall be used as group addresses for multicasting. Both the primary network and the secondary network of Vnet/IP network require two group addresses: one for multicasting to all Vnet/IP stations in the domain, and another for multicasting to all Vnet/IP stations in the Vnet/IP network.

To assign these group addresses to the stations, AD-HOC Block group address 224.0.23.33 may be used.

NOTE This group address is registered with the Internet Assigned Numbers Authority (IANA).

6.4.2 IGMP

IGMP shall be used to perform multicasting.

6.4.3 TOS

The following five TOS parameter values shall be used for the Data Link service:

a) time synchronization
b) high priority
c) medium priority
d) low priority
e) general purpose

6.5 Transport Layer

Internet standard RFC 768 (UDP, User Datagram Protocol) and its amendments and successors, and the Data Link Layer specified in Sections 4 and 5 of this PAS shall be used.

6.5.1 Port No

UDP port number 5313 shall be used.

NOTE This UDP port number is registered with the Internet Assigned Numbers Authority (IANA).

6.5.2 Parameter selection

Table 6 shows the parameter selection.

Table 6 – Transport Layer Parameter selection

Parameter Symbol	Parameter name	Usage	Value
P(ND)	max-domains	M	31
P(HC)	max-hop-count	M	7
P(NS)	max-stations	M	64
$P(GA_{1A})$	IP-group-address-1A	M	239.192.24.0
$P(GA_{1B})$	IP-group-address-1B	M	239.192.24.1
$P(GA_{2A})$	IP-group-address-2A	M	239.192.24.4
$P(GA_{2B})$	IP-group-address-2B	M	239.192.24.5
$P(AB_{DA})$	IP-base-address-domain-A	M	192.168.0.0
$P(AB_{DB})$	IP-base-address-domain-B	M	192.268.128.0
$P(MRC_{AUS})$	max-retry-count-AUS	M	5
$P(MRC_{ASS})$	max-retry-count-ASS	M	5
P(MOS)	max-outstanding-number	M	10
$P(TNR_{AUS})$	max-response-time-AUS	M	50 ms
$P(TNR_{ASS})$	max-response-time-ASS	M	500 ms
$P(TWT_{AUS})$	wait-time-AUS	M	100 ms
$P(TWT_{ASS})$	wait-time-ASS	M	100 ms
$P(TID_{ASS})$	inter-DTPDU-time	M	500 ms
P(MC)	macro-cycle-period	M	100 ms
$P(SD_{UUS})$	starting-delay-UUS	M	{5,55} ms
$P(SD_{AUS})$	starting-delay-AUS	M	{5,55} ms
$P(SD_{ASS})$	starting-delay-ASS	M	{10,20,30,40,60,70,80} ms
$P(SD_{MUS})$	starting-delay-MUS	M	{12,32,62,82} ms
$P(SD_{MSS})$	starting-delay-MSS	M	{40} ms
$P(TD_{UUS})$	time-duration-UUS	M	35 ms
$P(TD_{AUS})$	time-duration-AUS	M	35 ms
$P(TD_{ASS})$	time-duration-ASS	M	1 ms
$P(TD_{MUS})$	time-duration-MUS	M	1 ms
$P(TD_{MSS})$	time-duration-MSS	M	5 ms
$P(TO_{UUS})$	offset-time-UUS	M	1 ms
$P(TO_{AUS})$	offset-time-AUS	M	1 ms
$P(TO_{ASS})$	offset-time-ASS	M	1 ms
$P(TO_{MUS})$	offset-time-MUS	M	1 ms
$P(TO_{MSS})$	offset-time-MSS	M	1 ms
$P(DV_{UUS})$	divisor-for-grouping	M	5
$P(DV_{AUS})$	divisor-for-grouping	M	5
$P(DV_{ASS})$	divisor-for-grouping	M	8
$P(DV_{MUS})$	divisor-for-grouping	M	8
$P(DV_{MSS})$	divisor-for-grouping	M	1
P(KS)	key size	M	4
P(AS)	authentication field size	M	1
P(PN)	prime-number	M	0b01011001
P(BS)	base-number	M	251
P(UD)	key-update-time	M	3600 s

6.6 Application Layer

The Application Layer specified in Parts 2 and 3 of this document shall be used. The specifications of Layers 5 and 6 are included in the Application Layer's specifications of this document.

Section 2: Application Layer Service definition

7 Concepts

7.1 General

This Fieldbus Application Layer (FAL) provides communication services to time-critical and nontime-critical applications in fieldbus devices.

7.2 Relationships between ASEs

7.3 FAL ASEs

A modular approach was taken in the definition of FAL ASEs. The ASEs defined for the FAL are also object-oriented. In general, ASEs provide a set of services designed for one specific object class or for a related set of classes.

To support remote access to the AP, the Application Relationship ASE is defined. This application relationship ASE provides services to the AP for defining and establishing communication relationships with other APs, and provides services to other ASEs for conveying their service requests and responses.

Each FAL ASE defines a set of services, APDUs and procedures that operate on the classes that it represents. Only a subset of the ASE services may be provided to meet the needs of an application. Profiles may be used to define such subsets.

APDUs are sent and received between FAL ASEs that support the same services. Figure 1 shows the FAL ASEs from the perspective of communication.

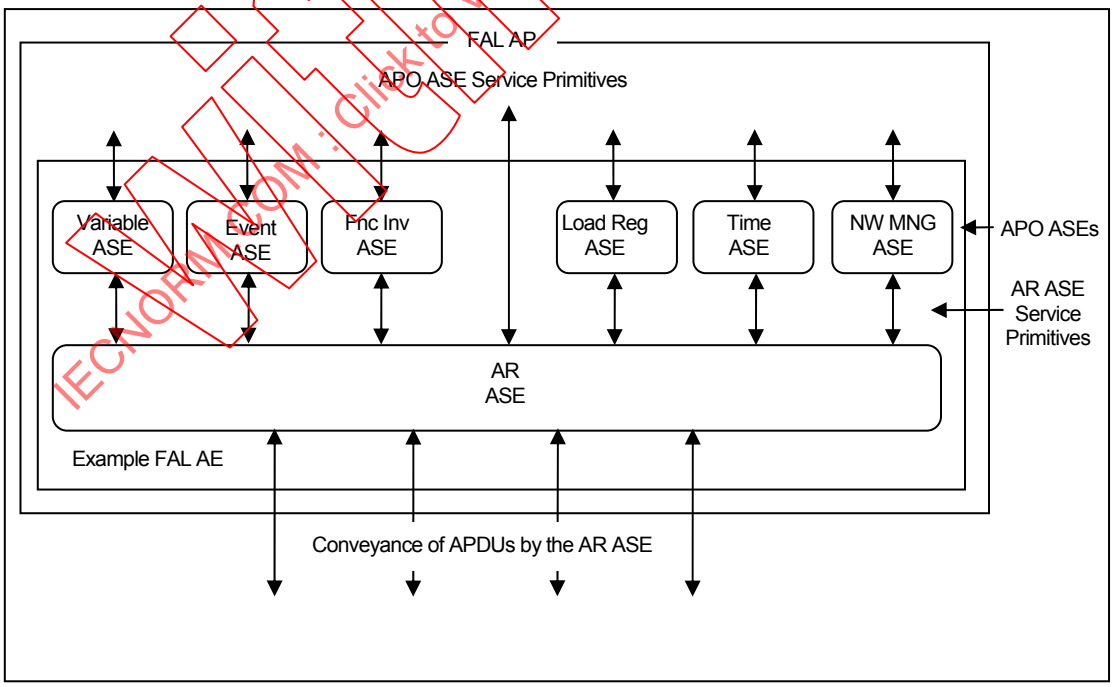

Figure 1 – FAL ASEs

7.4 Common FAL service parameters

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

Argument
This parameter carries the parameters of the service invocation.

AREP
This parameter contains information sufficient for local identification of the AREP to be used to convey the service. This parameter may use a key attribute of the AREP to identify the application relationship. When an AREP supports multiple contexts simultaneously, the AREP parameter is extended to identify the context as well as the AREP.

InvokeID
This parameter identifies this invocation of the service. InvokeID is used to associate service requests with responses. Therefore, no two outstanding service invocations can be identified by the same InvokeID value.

Result(+)
This selection type parameter indicates that the service request was successful.

Result(-)
This selection type parameter indicates that the request failed.

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

8 ASEs

8.1 Variable ASE

8.1.1 Overview

In the fieldbus environment, application processes contain data that can be both read and written by remote applications. The variable ASE defines the network-visible attributes of application data and provides a set of services used to read, write and report their values.

When the appropriate types of application relationship are supported, the variable ASE may be used to support two different access models i.e., the client/server model and the publisher/subscriber model. The client/server model is characterized by a client application sending a read or write request to a server application that responds accordingly. The server's activity is stimulated by clients on the network; if there are no requests, the server generates no responses.

The publisher/subscriber model is different in that it is characterized by a data producer publishing its data onto the network. Subscribers wishing to acquire the published data join the application relationship used to publish it and listen for the data as it is transmitted. Two models are provided to support this publisher/subscriber activity, i.e., the “pull” model and the “push” model.

In the “pull” model, one of the subscribers requests that the publisher publishes a sequence of variable data by issuing a publish request to it. The publisher distributes the variable data periodically according to the remote request by multicasting. The publishing schedule is controlled by the publisher itself.

In the “push” model, the publisher is requested to distribute a sequence of variable data by the local FAL user. The publisher distributes a sequence of variable data by multicasting

according to the local request. The publishing schedule is also controlled by the publisher itself.

8.1.2 Variable class specification

8.1.2.1 Formal model

FAL ASE:			**VARIABLE ASE**
CLASS:			**VARIABLE**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Key Attribute:	Numeric Identifier
2	(m)	Attribute:	Data Type
3	(0)	Attribute:	Length
4	(o)	Attribute:	Read enable
5	(o)	Attribute:	Write enable
SERVICES:			
1	(o)	OpsService:	Read
2	(o)	OpsService:	Write
3	(o)	OpsService:	Information Report

8.1.2.2 Attributes

Numeric Identifier
This key attribute identifies an instance of this object class.

Data Type
This attribute is the numeric identifier of the data type.

Length
This optional attribute is the length of the data in octets.

Read enable
The value of this optional attribute indicates whether the data value of the Variable Object can be read via the fieldbus.

Write enable
The value of this optional attribute indicates whether the data value of the Variable Object can be updated via the fieldbus.

8.1.3 Variable List class specification

8.1.3.1 Formal model

FAL ASE:			**VARIABLE ASE**
CLASS:			**VARIABLE LIST**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Key Attribute:	Numeric Identifier
2	(m)	Attribute:	Number of Entries
3	(m)	Attribute:	List Of Variables
SERVICES:			
1	(o)	OpsService:	Read
2	(o)	OpsService:	Write
3	(o)	OpsService:	Information Report

8.1.3.2 Attributes

Numeric Identifier
This key attribute identifies an instance of this object class.

Number of Entries
This attribute contains the number of variables in the list.

List of Variables
This attribute identifies the variables by key attribute that are contained in the list.

8.1.4 Variable ASE service specification

8.1.4.1 Supported services

This subclause contains the definition of services that are unique to this ASE. The services defined for this ASE are:

- Read
- Write
- Information Report

8.1.4.2 Read service

8.1.4.2.1 Service overview

This confirmed service may be used to read the value of a variable object or a variable list object. It is not used with the publisher/subscriber model.

8.1.4.2.2 Service primitives

The service parameters for each primitive are shown in Table 7.

Table 7 – Read service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
AREP	M	M(=)		
InvokeID	M	M(=)		
Variable Specifier	M	M(=)		
Result(+)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Value			M	M(=)
Result(-)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Error Info			M	M(=)

Variable Specifier
This parameter specifies a variable object or a variable list object to be read by the key attribute.

Value
This parameter contains the value read. For each of the variables, this parameter contains the value of the variable. For variable lists, this parameter contains the values of each of the variables in the list concatenated together in the order in which they appear in the list.

NOTE If any of the variables in a variable list could not be read, the service fails.

8.1.4.3 Write service

8.1.4.3.1 Service overview

This confirmed service is used to write the value of a variable object or a variable list object. It is not used with the publisher/subscriber model.

8.1.4.3.2 Service primitives

The service parameters for each primitive are shown in Table 8.

Table 8 – Write service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
AREP	M	M(=)		
InvokeID	M	M(=)		
Variable Specifier	M	M(=)		
Value	M	M(=)		
Result(+)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Result(-)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Error Info			M	M(=)

Variable Specifier
This parameter specifies a variable object or a variable list object to be written by the key attribute.

Value
This parameter contains the value to write. For variables, this parameter contains the value of the variable. For variable lists, this parameter contains the values of each of the variables in the list concatenated together in the order that they appear in the list.

NOTE If any variable in a variable list object cannot be updated, none of the variables in the variable list object will be updated and the write will fail.

8.1.4.4 Information Report service

8.1.4.4.1 Service overview

This optional service is an unconfirmed service that may be used to report the value of a variable object or a variable list object. This service may be used in the publisher/subscriber push model.

8.1.4.4.2 Service primitives

The service parameters for each primitive are shown in Table 9.

Table 9 – Information Report service parameters

Parameter name	Req	Ind
Argument	M	M(=)
AREP	M	M(=)
Variable Specifier	M	M(=)
Value	M	M(=)

Variable Specifier
This parameter specifies a variable object or a variable list object to be reported by the key attribute.

Value
This parameter contains the value to be reported. For variables, this parameter contains the value of the variable. For variable lists, this parameter contains the value of each variable in the list concatenated together in the order that they appear in the list.

NOTE If any of the variables in a variable list could not be read, the service fails.

8.2 Event ASE

8.2.1 Overview

Event objects are used to define messages reporting event occurrences. Event messages contain information that identifies and describes event occurrences.

Notifiers are responsible for collecting event messages from event objects, and distributing one or more event message(s) in a single invocation of the FAL event notification service. The number of event messages that may be submitted in a single service invocation is limited by the maximum APDU size that can be transferred by the AR.

If an application process fails to receive one or more event notifications, a notification recovery service is provided to request the notifier to retransmit the notifications.

In this model, application processes are responsible for providing the functions for event objects and event list objects, and the FAL is responsible for providing communication services designed specifically for them. The application process detects events, builds event messages and aggregates them together. The application process distributes the aggregated event messages using the FAL event notification service.

8.2.2 Event class specification

8.2.2.1 Formal model

FAL ASE:			**EVENT ASE**
CLASS:			**NOTIFIER**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Key Attribute:	Numeric Identifier
2	(m)	Attribute:	AREP
3	(o)	Attribute:	Last Notification Sequence Number
4	(o)	Attribute:	List Of Events
SERVICES:			
1	(o)	OpsService:	Event Notification
2	(o)	OpsService:	Notification Recovery

8.2.2.2 Attributes

8.2.2.2.1 Numeric Identifier

This key attribute identifies an instance of this object class.

8.2.2.2.2 AREP

This attribute identifies the AREP configured to convey event notifications. This AREP is also used for reporting the event notifications generated by an event recovery request.

8.2.2.2.3 Last Notification Sequence Number

The conditional attribute contains the last sequence number used. It is incremented for each event notification service invocation.

8.2.2.2.4 List of Events

This optional attribute identifies the events that are configured.

8.2.3 Event ASE service specification

8.2.3.1 Supported services

This subclause contains the definition of services that are unique to this ASE. The services defined for this ASE are:

- Event Notification
- Notification Recovery

8.2.3.2 Event Notification service

8.2.3.2.1 Service overview

This unconfirmed service is used by the notifier of an FAL AP to notify other APs that one or more events have occurred.

8.2.3.2.2 Service primitives

The service parameters for each primitive are shown in Table 10.

Table 10 – Event Notification service parameters

Parameter name	Req	Ind
Argument	M	M(=)
AREP	M	M(=)
NotifierID	M	M(=)
Sequence Number	U	U(=)
Notification Time	U	U(=)
List of Event Messages	M	M(=)
Event Key Attribute	M	M(=)
Event Data Type	C	C(=)
Event Detection Time	U	U(=)
Event Data	U	U(=)

NotifierID
This parameter identifies the notifier issuing the event notification.

Sequence Number
This optional parameter is the sequence number for the event notification. It may be used for notification recovery purposes.

Notification Time
This optional parameter is the time tag for event notification.

List of Event Messages
This parameter contains the list of event messages that are to be reported. It may contain messages from one or more event objects. The contents of each message are specified by its event object and shall be consistent with that specified for the notifier object.

Event Key Attribute
This parameter identifies each of the specific events being acknowledged by this service.

Event Data Type
This conditional parameter indicates the data type of each of the event data parameters. This parameter may be present only if the event data parameter is present. If the event data parameter is present, this parameter shall be present.

Event Detection Time
This optional parameter reports the time of the event detection. This parameter is present only if it is defined for the specified event object and is supported by the specified notifier.

Event Data
This optional parameter contains user data to be included in an event message in addition to that used to identify the event occurrence. This parameter is present only if it is defined for the specified event object and is supported by the specified notifier.

8.2.3.3 Notification Recovery service

8.2.3.3.1 Service overview

This unconfirmed service is used to request that a specified number of retained event notifications be returned. Notifications are returned using the event notification service.

8.2.3.3.2 Service primitives

The service parameters for each primitive are shown in Table 11.

Table 11 – Event Notification Recovery service parameters

Parameter name	Req	Ind
Argument	M	M(=)
AREP	M	M(=)
NotifierID	M	M(=)
Sequence Number	U	U(=)

NotifierID
This parameter identifies the notifier to which this service is directed.

Sequence Number
This optional parameter specifies the sequence number of the event notification to be re-sent. If not present, the last notification sent is being requested.

8.3 Load Region ASE

8.3.1 Overview

A load region represents an unstructured memory area whose contents is to be uploaded (read) or downloaded (written). In this context, “unstructured” means that the memory area is represented only as an ordered sequence of octets. No other structure is apparent.

A load region may represent an unnamed volatile memory area, such as that implemented by dynamic computer memory, or a named non-volatile memory object, such as a file. The contents of a load region are referred to as a load image and can contain programs or data. The transfer of a load image to or from a load region is performed using the load process.

This load region model provides services that permit an application process to initiate the downloading or uploading of specified load regions.

8.3.2 Load Region class specification

8.3.2.1 Formal model

FAL ASE:			**LOAD REGION ASE**
CLASS:			**LOAD REGION**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Attribute:	Load Region Size
2	(m)	Attribute:	Local Address
3	(o)	Attribute:	Load Image Name
SERVICES:			
1	(m)	OpsService:	Download Services
2	(m)	OpsService:	Upload Services

8.3.2.2 Attributes

Load Region Size
This attribute specifies the maximum size of the load region in octets.

Local Address
This attribute is a locally significant address of the load region.

Load Image Name
This optional attribute specifies the name of the load image contained in the load region.

8.3.3 Load Region ASE service specification

8.3.3.1 Supported services

This subclause contains the definitions of services that are unique to this ASE. The services defined for this ASE are:

- Download
- Upload

8.3.3.2 Download service

8.3.3.2.1 Service overview

This confirmed service is used to download a load image in its request and indication primitives. The response and confirmation primitives are used to convey the success or failure of the download.

8.3.3.2.2 Service primitives

The service parameters for each primitive are shown in Table 12.

Table 12 – Download service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
AREP	M	M(=)		
InvokeID	M	M(=)		
Load Region	M	M(=)		
Load Data	M	M(=)		
Result(+)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Result(-)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Error Info			M	M(=)

Load Region
This parameter specifies the load region into which the image is to be downloaded.

Load Data
This parameter contains the data to be downloaded.

8.3.3.3 Upload service

8.3.3.3.1 Service overview

This confirmed service is used to upload a load image. Its request and indication primitives convey an upload request. The response and confirmation primitives are used to convey the load image of the load region and the result of loading.

8.3.3.3.2 Service primitives

The service parameters for each primitive are shown in Table 13.

Table 13 – Upload service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
AREP	M	M(=)		
InvokeID	M	M(=)		
Load Region	M	M(=)		
Result(+)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Load Data			M	M(=)
Result(-)			S	S(=)
AREP			M	M(=)
InvokeID			M	M(=)
Error Info			M	M(=)

Load Region
This parameter specifies the load region from which the image is to be uploaded.

Load Data
This parameter contains the data uploaded.

8.4 Function Invocation ASE

8.4.1 Overview

The function invocation class models the state-oriented function invocation. It may be used to model software processes or user functions the operation of which may be controlled.

8.4.2 Function Invocation class specification

8.4.2.1 Formal model

FAL ASE:			**FUNCTION INVOCATION ASE**
CLASS:			**FUNCTION INVOCATION**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Key Attribute:	Identifier
2	(m)	Attribute:	Function Invocation State
SERVICES:			
1	(o)	OpsService:	Start
2	(o)	OpsService:	Stop
3	(o)	OpsService:	Resume

8.4.2.2 Attributes

8.4.2.2.1 Identifier

This key attribute consists of a station identifier and a function identifier.

8.4.2.2.2 Function Invocation State

This attribute indicates the current state of the function invocation. An enumerated set of values has been defined.

UNRUNNABLE	This state indicates that the function invocation is not executing and can not be executed.
IDLE	This state indicates that the function invocation is not executing, but is capable of being executed.
RUNNING	This state indicates that the function invocation is executing.
STOPPED	This state indicates that the execution of a function invocation has been suspended.

8.4.3 Function Invocation ASE service specification

8.4.3.1 Supported services

This subclause contains the definition of services that are unique to this ASE. The services defined for this ASE are:

- Start
- Stop
- Resume

8.4.3.2 Start service

8.4.3.2.1 Service overview

This confirmed service is used to start a function invocation from the initial condition.

8.4.3.2.2 Service primitives

The service parameters for each primitive are shown in Table 14.

Table 14 – Start service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	=(M)		
AREP	M	=(M)		
InvokeID	M	=(M)		
FunctionID	M	=(M)		
Result			M	=(M)
AREP			M	=(M)
InvokeID			M	=(M)
Error Info			M	=(M)

FunctionID
This parameter specifies one of the key attributes of the function invocation object.

8.4.3.3 Stop service

8.4.3.3.1 Service overview

This confirmed service is used to stop a function invocation retaining its context so that it may be resumed.

8.4.3.3.2 Service primitives

The service parameters for each primitive are shown in Table 15.

Table 15 – Stop service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	=(M)		
AREP	M	=(M)		
InvokeID	M	=(M)		
FunctionID	M	=(M)		
Result			M	=(M)
AREP			M	=(M)
InvokeID			M	=(M)
Error Info			M	=(M)

FunctionID
This parameter specifies one of the key attributes of the function invocation object.

8.4.3.4 Resume service

8.4.3.4.1 Service overview

This confirmed service is used to request to start a function invocation from the suspended condition.

8.4.3.4.2 Service primitives

The service parameters for this service are shown in Table 16.

Table 16 – Resume service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	=(M)		
AREP	M	=(M)		
InvokeID	M	=(M)		
FunctionID	M	=(M)		
Result			M	=(M)
AREP			M	=(M)
InvokeID			M	=(M)
Error Info			M	=(M)

FunctionID
This parameter specifies one of the key attributes of the function invocation object.

8.5 Time ASE

8.5.1 Overview

The Time ASE specifies the clock synchronization mechanism to synchronize the time among network stations. This time can be used to add a time value to alert information and to sequence messages in a time-wise order.

A system consists of one or more time master(s) and several time slaves. Clock synchronization is performed by the communication system. Adjustment and management of the clock are tasks of the application.

The network shall have at least one network time master which may be one of the domain time masters selected automatically or may be a fixed station.

The network time master has the master clock of the network. The master clock may be the local clock of the network time master itself or it may be synchronized with a clock source outside the network.

The domain time master of each domain obtains network time from the network time master and it distributes the network time value to stations in the domain.

8.5.2 Time class specification

8.5.2.1 Formal model

FAL ASE:			**Time ASE**
CLASS:			**Time**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Attribute:	Implicit
2	(m)	Attribute:	Role
3	(o)	Attribute:	Stratum
4	(o)	Attribute:	Poll Interval
5	(o)	Attribute:	Precision
SERVICES:			
1	(m)	OpsService:	Get Network Time
2	(m)	OpsService:	Set Network Time
3	(m)	OpsService:	Tick Notification service

8.5.2.2 Attributes

8.5.2.2.1 Role

This attribute specifies the role of the Time ASE with values defined as follows:

NETWORK TIME MASTER A network unique end node which responds to time requests from domain masters.

DOMAIN TIME MASTER A domain unique end node which distributes time information within the domain.

TIME SLAVE An end node subscribes time information distributed by the domain master.

8.5.2.2.2 Stratum

This attribute indicates the stratum level of the local clock.

8.5.2.2.3 Poll Interval

This attribute indicates the maximum interval between successive time messages.

8.5.2.2.4 Precision

This attribute indicates the precision level of the local clock.

8.5.3 Time ASE service specification

8.5.3.1 Supported services

This subclause contains the definitions of services that are unique to this ASE. The services defined for this ASE are:

- Get Network Time
- Set Network Time
- Tick Notification

8.5.3.2 Get Network Time service

8.5.3.2.1 Service overview

This local service shall be used to obtain the network time value that is distributed from the network. If Time ASE is the network time master, the local time value of the master clock is returned.

8.5.3.2.2 Service primitives

The service parameters for each primitive are shown in Table 17.

Table 17 – Get Network Time service parameters

Parameter name	Req	Cnf
Argument	M	
InvokeID	M	
Result		M
InvokeID		M
Network Time		M
Status		M
Error info		C

Network Time
This parameter is the value of the network time.

Status
This parameter indicates the states of time synchronization. The possible states are;

- Not synchronized
- Synchronized to the domain time master
- Synchronized to the network time master as the domain time master
- Synchronized to an external time source as a network time master

8.5.3.3 Set Network Time service

8.5.3.3.1 Service overview

This confirmed service shall be used to set the value of network time to the time master.

This service is available under conditions where the network time master is not synchronized to an external time source.

8.5.3.3.2 Service primitives

The service parameters for each primitive are shown in Table 18.

Table 18 – Set Network Time service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
InvokeID	M	M(=)		
Network Time	M	M(=)		
Result			M	M(=)
InvokeID			M	M(=)
Error Info			C	C(=)

Network Time
This parameter is the value of the network time to be set.

8.5.3.4 Tick Notification service

8.5.3.4.1 Service overview

This local service shall be used to recognize tick timing synchronized to the network time. The tick interval is configurable.

8.5.3.4.2 Service primitives

The service parameters for each primitive are shown in Table 19.

Table 19 – Set Network Time service parameters

Parameter name	Ind
Result	M
Tick	M

Tick
This parameter indicates tick timing.

NOTE This service may be implemented by means of a hard-wired interruption.

8.6 Network Management ASE

8.6.1 Overview

End nodes may have two network interfaces that provide network redundancy. These interfaces are referred to as network interface A and network interface B. The former shall be connected to the primary network, while the latter shall be connected to the secondary network.

Each Network Management ASE constructs two diagnostic message APDUs and sends them concurrently to network interfaces A and B. Each Network Management ASE receives pairs of APDUs from other end nodes on the network that participate in network redundancy. This information is used to determine which network interface (A or B) is to be selected to send other APDUs. This information is also used to determine whether the destination node is reachable in advance of sending an APDU.

This information is preserved in the network status table.

8.6.2 Network Management class specification

8.6.2.1 Formal model

FAL ASE:			**Network Management**
CLASS:			**Network Management**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Attribute:	Number of Network Interfaces
2	(m)	Attribute:	Diagnostic Message Interval
3	(m)	Attribute:	Aging Time
4	(m)	Attribute:	List of Network Interface Status
SERVICES:			
1	(m)	OpsService:	Get Network Status
2	(m)	OpsService:	Get Station Status
3	(m)	OpsService:	Network Status Change Report
4	(m)	OpsService:	Station Status Change Report

8.6.2.2 Attributes

8.6.2.2.1 Number of Network Interfaces

This attribute specifies the number of network interfaces on this end node. An end node may have one or two network interfaces.

8.6.2.2.2 Diagnostic Message Interval

This attribute specifies the time interval between successive invocations of the diagnostic messages in milliseconds.

8.6.2.2.3 Aging Time

This attribute specifies the time interval used to remove silent end nodes from the List of Network Interface Status.

8.6.2.2.4 List of Network Interface Status

This attribute is a list of the path status representing the condition of the path to a remote station from which the Network Management ASE may be receiving successive Diagnostic Messages. The status is bi-directional for a path and indicates whether Diagnostic Messages are being received successfully at each end of the path or whether either end of the path is in a fault state.

8.6.3 Network Management ASE service specification

8.6.3.1 Supported services

This subclause contains the definition of the services that is unique to this ASE.

The services defined for this ASE are:

- Get Network Status
- Get Station Status
- Network Status Change Report
- Station Status Change Report

8.6.3.2 Get Network Status service

8.6.3.2.1 Service overview

This local service is used to obtain the status of the network.

8.6.3.2.2 Service primitives

The service parameters for each primitive are shown in Table 20.

Table 20 – Get Network Status service parameters

Parameter name	Req	Cnf
Argument	M	
InvokeID	M	
Result		M
InvokeID		M
Network Status		M

Network Status
This parameter indicates consistency of the primary and secondary networks. Possible values are:

- HEALTHY
- FAILED

8.6.3.3 Get Station Status service

8.6.3.3.1 Service overview

This local service is used to obtain the status of the specified station.

8.6.3.3.2 Service primitives

The service parameters for each primitive are shown in Table 21.

Table 21 – Get Station Status service parameters

Parameter name	Req	Cnf
Argument	M	
Station Identifier	M	
InvokeID	M	
Result		M
InvokeID		M
Station Status		M
Route Status		M

Station Identifier
This parameter indicates the station for which the status is requested, and consists of a domain number and a station number.

Station Status
This parameter indicates the status of the station specified by the station identifier, and includes the following information.

a) existence

TRUE	the station exists
FALSE	the station does not exist

b) redundancy

SINGLE	the station is not redundant
DUAL-REDUNDANT	the station is redundant

c) AREP of the end node which is on service

Route Status
This parameter indicates the status of route for the station, and includes the following information.

a) status of the route on the primary network

REACHABLE

UNREACHABLE

b) status of the route on the secondary network

REACHABLE

UNREACHABLE

8.6.3.4 Network Status Change Report service

8.6.3.4.1 Service overview

This local service is used to inform of changes in network status.

8.6.3.4.2 Service primitives

The service parameters for each primitive are shown in Table 22.

Table 22 – Network Status Change Report service parameters

Parameter name	Ind
Result	M
Network Status	M

Network Status
This parameter indicates consistency of the primary and secondary networks, and has the following possible values:

- HEALTHY
- FAILED

8.6.3.5 Station Status Change Report service

8.6.3.5.1 Service overview

This local service is used to inform of changes in the station status.

8.6.3.5.2 Service primitives

The service parameters for each primitive are shown in Table 23.

Table 23 – Station Status Change Report service parameters

Parameter name	Ind
Result	M
Station Identifier	M
Station Status	M
Route Status	M

Station Identifier
This parameter indicates the station of which the status has changed.

Station Status
This parameter indicates the status of the end node specified in/by the request primitive and includes the following information:

a) existence

TRUE	the station exists
FALSE	the station does not exist

b) redundancy

SINGLE	the station is not redundant
DUAL-REDUNDANT	the station is redundant

c) AREP of the end node which is on service

Route Status
This parameter indicates the status of the route for the station, and includes the following information:

a) status of the route on the primary network

REACHABLE

UNREACHABLE

b) status of the route on the secondary network

REACHABLE

UNREACHABLE

8.7 Application Relationship ASE

8.7.1 Overview

8.7.1.1 General

In a distributed system, application processes communicate with each other by exchanging Application Layer messages across well-defined Application Layer communication channels.

These communication channels are modeled in the Fieldbus Application Layer as application relationships (ARs).

ARs are responsible for conveying messages between applications according to specific communication characteristics required by time-critical systems. Different combinations of these characteristics lead to the definitions of different types of ARs. The characteristics of ARs are defined formally as attributes of AR endpoint classes.

The messages that are conveyed by ARs are FAL service requests and responses. Each is submitted to the AR ASE for transfer by an FAL Application Service Element (ASE) that represents the class of the APO being accessed. Figure 2 illustrates this concept.

Depending on the type of AR, APDUs may be sent to one or more destination application processes connected by the AR. Other characteristics of the AR determine how APDUs are to be transferred. These characteristics are described below.

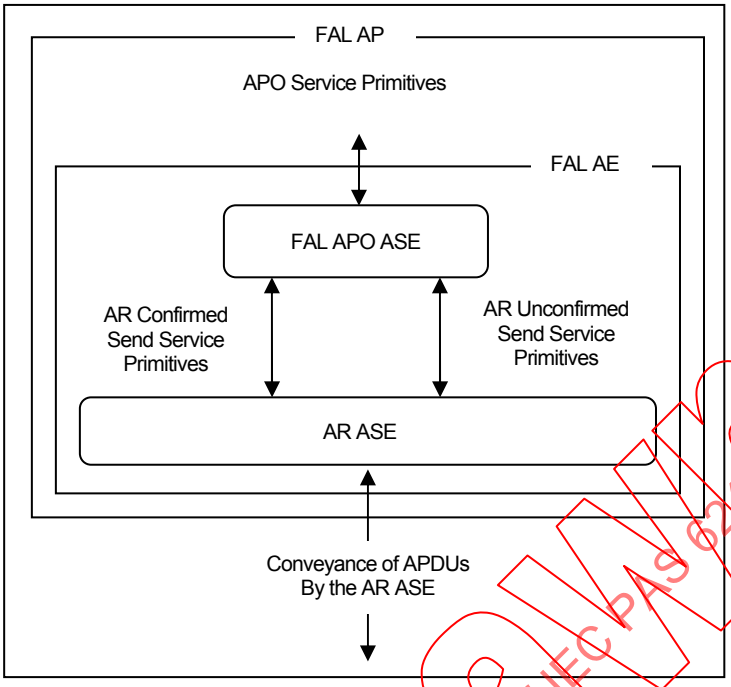

Figure 2 – The AR ASE conveys APDUs between APs

8.7.1.2 Endpoint context

Each AP involved in an AR contains an endpoint of the AR. Each AR endpoint is defined within the AE of the AP. An endpoint definition, when combined with the definitions of the other endpoints, defines an AR. To ensure communication compatibility among or between endpoints, each endpoint definition contains a set of compatibility-related characteristics.

These characteristics need to be configured appropriately for each endpoint for the AR to operate properly.

Endpoint definitions also contain a set of characteristics that describe the operation of the AR. These characteristics, when combined with those used to specify compatibility, define the context of the endpoint. The endpoint context is used by the AR ASE to manage the operation of the endpoint and the conveyance of APDUs. The characteristics that comprise the endpoint context are described in the next section.

8.7.1.2.1 Endpoint role

The role of an AREP determines the permissible behaviour of an AP at the AREP. An AREP role may be that of a client, server, peer (client and/or server), publisher or subscriber.

Table 24 and Table 25 summarize the characteristics and combinations of each of the AREP roles.

Table 24 – Conveyance of service primitives by AREP role

	Client	Server	Peer	Publisher	Subscriber
Send Service Req	X		X	X	
Recv Service Req		X	X		X
Send Service Rsp		X	X		
Recv Service Rsp	X		X		

Table 25 – Valid combinations of AREP roles involved in an AR

	Client	Server	Peer	Publisher	Subscriber
Client		X	X		
Server	X		X		
Peer	X	X	X		
Publisher					X
Subscriber				X	

8.7.1.2.2 Dedicated AR endpoints

Dedicated AR endpoints provide their services directly to the FAL User. Although their behaviour is the same as that of non-dedicated endpoints, the APDUs they convey contain no service specific protocol control information other than the AR control field.

FAL Users configured for dedicated ARs construct and transfer their data without using the services of the other FAL ASEs. The format and contents of the user data conveyed over dedicated ARs are known only through configuration.

8.7.1.2.3 Cardinality

The cardinality of an AR specifies, from the point of view of a client or publisher endpoint, how many remote application processes are involved in an AR. Cardinality is never expressed from the viewpoint of a server or a subscriber.

When expressed from the viewpoint of a client or peer endpoint, ARs are always one-to-one. Clients are never capable of issuing a request to multiple servers and waiting for responses from them.

When expressed from the viewpoint of a publisher endpoint, the cardinality indicates that multiple subscribers are supported. These one-to-many ARs provide communications between one application and a group of (one or more) applications. They are often referred to as multi-party and multicast.

In one-to-many ARs, the publisher endpoint identifies the remote endpoints using a group identifier, rather than identifying each one individually as is done with one-to-one ARs. This permits subscribers to join and leave ARs without disrupting the publisher endpoint context because their individual identities are not known to the publisher endpoint. However, the publisher application may be aware of their participation, but that information is not part of the AR endpoint context.

8.7.1.2.4 Conveyance model

The conveyance model defines how APDUs are sent between endpoints of an AR. Three characteristics are used to define these transfers:

- conveyance paths,
- trigger policy, and
- conveyance policy.

Conveyance paths

The purpose of AR ASEs is to transfer information between AR endpoints. This information transfer occurs over the conveyance paths of an AR. A conveyance path represents a one-way communication path used by an endpoint for input or output.

To support the role of the application process, the endpoint is configured with either one or two conveyance paths. Endpoints that only send or only receive are configured with either a send or receive conveyance path, respectively, and those that do both are configured with both. ARs with a single conveyance path are called unidirectional, and those with two conveyance paths are called bi-directional.

Unidirectional ARs are capable of conveying service requests only. To convey service responses, a bi-directional AR is necessary. Therefore, unidirectional ARs support the transfer of unconfirmed services in one direction only, while bi-directional ARs support the transfer of unconfirmed and confirmed services initiated by only one endpoint, or by both endpoints.

Trigger policy
Trigger policy indicates when APDUs are transmitted over the network by the Data Link Layer.

The first type is referred to as user-triggered. User-triggered AREPs submit FAL APDUs to the Data Link Layer for transmission at the earliest opportunity.

The second type is referred to as network-scheduled. Network scheduled AREPs submit FAL APDUs to the Data Link Layer for transmission according to a schedule configured by management.

Conveyance policy
Conveyance policy indicates whether APDUs are transferred according to a buffer model or a queue model. These models describe the method of conveying APDUs from the sender to the receiver.

Buffered ARs contain conveyance paths that have a single buffer at each endpoint. Updates to the source buffer are conveyed to the destination buffer according to the trigger policy of the AR. Updates to either buffer replace its contents with new data. In buffered ARs, unconveyed or undelivered data that have been replaced are lost. In additional data contained in a buffer may be read multiple times without destroying its contents.

Queued ARs contain conveyance paths that are represented as a queue between endpoints. Queued ARs convey data using a FIFO queue. Queued ARs are not overwritten, meaning that new entries are queued until they can be conveyed and delivered.

If a queue is full, a new message will not be placed into the queue.

8.7.1.3 AR establishment

For an AR endpoint to be used by an application process, the corresponding AR shall be active. When an AR is activated, it is referred to as “established”.

AR establishment can occur in one of three ways.

First, ARs can be pre-established, meaning that the AE that maintains the endpoint context is created when the AP is connected to the network. In this case, communications among the applications involved in the AR may take place without first having to explicitly establish the AR. Any AR may be defined as being pre-established.

Second, ARs can be pre-defined, but not pre-established. Pre-defined means that the characteristics of the AR are known to each endpoint, but that their contexts have not been synchronized for operation. In this case, the use of the FAL Establish service is required to synchronize them and open them for data transfer.

Third, ARs can require dynamic definition and establishment. In this case, definitions shall be created for each of the AREPs using the Create service. After creation, they are established using the Establish service if they were not created as “established”.

8.7.2 Application Relationship Class specification

8.7.2.1 Formal model

The AR endpoint formal model defines the characteristics common to all AR endpoints. This class is not capable of being instantiated. It is present only for the inheritance of its attributes and services by its subclasses, each specified in a separate subclause of this PAS.

All AR endpoint attributes are accessible through system Management.

FAL ASE:			**AR ASE**
CLASS:			**AR ENDPOINT**
CLASS ID:			**not used**
PARENT CLASS:			**TOP**
ATTRIBUTES:			
1	(m)	Attribute:	FAL Revision
2	(m)	Attribute:	Dedicated (TRUE, FALSE)
3	(m)	Attribute:	Cardinality (one-to-one, one-to-many)
4	(m)	Attribute:	Conveyance policy (Queued, Buffered)
5	(m)	Attribute:	Conveyance path (Unidirectional, Bi-directional)
6	(m)	Attribute:	Trigger policy (User-triggered, Network-scheduled)
7	(o)	Attribute:	Transfer Syntax
SERVICES:			
1	(o)	OpsService:	AR-Unconfirmed Send
2	(o)	OpsService:	AR-Confirmed Send
3	(o)	OpsService:	AR-Establish
4	(o)	OpsService:	AR-Abort

8.7.2.2 Attributes

FAL Revision
This specifies the revision level of the FAL protocol used by this endpoint. The revision level is in the AR header of all FAL-PDUs transmitted.

Dedicated
This attribute specifies whether the endpoint is dedicated or not. When TRUE, the services of the AR ASE are accessed directly by the FAL User.

Cardinality
This attribute specifies cardinality of the AR described in subclause 2.7.1.

Conveyance policy
This attribute specifies conveyance policy of the AR described in subclause 2.7.1.

Conveyance path
This attribute specifies conveyance path of the AR described in subclause 2.7.1.

Trigger policy
This attribute specifies trigger policy of the AR described in subclause 2.7.1.

Transfer Syntax
This optional attribute identifies the encoding rules to be used on the AR. When not present, the default FAL Transfer Syntax of this standard is used.

8.7.2.3 Services

All services defined for this class are optional. When an instance of the class is defined, at least one shall be selected.

AR-Unconfirmed Send
This optional service is used to send an unconfirmed service.

AR-Confirmed Send
This optional service is used to send a confirmed service.

AR-Establish
This optional service is used to establish (open) an AR.

AR-Abort
This optional service is used to abort (abruptly terminate) an AR.

8.7.3 Application Relationship ASE service specification

8.7.3.1 Supported services

This subclause contains the definitions of services that are unique to this ASE. The services defined for this ASE are:

- AR-Unconfirmed Send
- AR-Confirmed Send
- AR-Establish
- AR-Abort

The services AR-Confirmed Send contain the FAL PDU body as part of the Result parameter in the response and confirmation primitives. The FAL PDU body may contain either the positive response or negative response returned by the FAL User transparently to the AR ASE. Therefore, these services have a single Result parameter instead of the separate Result(+) and Result(-) parameters that are commonly used to convey the positive and negative responses returned by the FAL User.

8.7.3.2 AR-Unconfirmed Send service

8.7.3.2.1 Service overview

This service is used to send AR-Unconfirmed Send request APDUs for FAL APO ASEs. The AR-Unconfirmed Send service may be requested at either endpoint of a one-to-one bi-directional AR, at the server endpoint of a one-to-one unidirectional AR, or at the publisher endpoint of a one-to-many AR.

8.7.3.2.2 Service primitives

The service parameters for each primitive are shown in Table 26.

Table 26 – AR-Unconfirmed Send

Parameter name	Req	Ind
Argument	M	M(=)
AREP	M	M(=)
Remote DL Address	M	M(=)
FAL Service Type	M	M(=)
FAL APDU Body	M	M(=)

Remote DL Address
This parameter contains the destination DLSAP address in the request and the source DLSAP address in the indication.

FAL Service Type
This parameter contains the type of service being conveyed.

FAL APDU Body
This parameter contains the service-dependent body for the APDU.

8.7.3.2.3 Service procedure

The AR-Unconfirmed Send service is a service that operates through a queue.

The requesting FAL ASE submits an AR-Unconfirmed Send request primitive to its AR ASE. The AR ASE builds an AR-Unconfirmed Send request APDU.

The AR ASE queues the APDU for submission to the lower layer.

If the AREP is user-triggered, the AR ASE immediately requests the lower layer to transfer the APDU. If the AR is network-scheduled, the AR ASE requests the DL to transfer the data at the scheduled time. The Data Link mapping indicates how the AR ASE coordinates its requests to transmit the data with the Data Link Layer.

Upon receipt of the AR-Unconfirmed Send request APDU, the receiving AR ASE delivers an AR-Unconfirmed Send indication primitive to the appropriate FAL ASE as indicated by the FAL Service Type Parameter.

8.7.3.3 AR-Confirmed Send service

8.7.3.3.1 Service overview

This service is used to send confirmed request and response APDUs for FAL APO ASEs. The AR-Confirmed Send service may be requested at Client and Peer endpoints of one-to-one bi-directional ARs.

8.7.3.3.2 Service primitives

The service parameters for each primitive are shown in Table 27.

Table 27 – AR-Confirmed Send

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
AREP	M	M(=)		
InvokeID	M	M(=)		
Server DL-Address	M	M(=)		
FAL Service Type	M	M(=)		
FAL APDU Body	M	M(=)		
Result			M	M(=)
AREP			M	M(=)
InvokeID			M	M(=)
Client DL-Address			M	M(=)
FAL Service Type			M	M(=)
FAL APDU Body			M	M(=)

Server DL-Address
This parameter contains the DL-address of the server.

FAL Service Type
This parameter contains the type of service being conveyed.

FAL APDU Body
This parameter contains the service-dependent body for the APDU.

Result
This parameter indicates that the service request succeeded or failed.

Client DL-Address
This parameter contains the DL-address of the client.

8.7.3.3.3 Service procedure

The AR-Confirmed Send service is a service that operates through a queue.

The requesting FAL ASE submits an AR-Confirmed Send request primitive to its AR ASE. The AR ASE creates a transaction state machine to control the invocation of the service.

The AR ASE builds an AR-Confirmed Send request APDU and queues the APDU for submission to the lower layer, then the AR ASE immediately requests the lower layer to transfer the APDU.

Upon receipt of the AR-Confirmed Send request APDU, the receiving AR ASE delivers an AR-Confirmed Send indication primitive to the appropriate FAL ASE as indicated by the FAL Service Type Parameter.

The responding FAL ASE submits a confirmed send response primitive to its AR ASE. The AR ASE builds a confirmed send response APDU.

The AR ASE queues the APDU for submission to the lower layer. In addition, the AR ASE immediately requests the lower layer to transfer the APDU.

Upon receipt of the confirmed send response APDU, the receiving AR ASE uses the InvokeID contained in the response APDU to associate the response with the appropriate request and cancel the associated transaction state machine. The AR ASE delivers an AR-Confirmed Send confirmation primitive to the requesting FAL ASE.

If the timer expires before the sending AR ASE receives the response APDU, the AR ASE cancels the associated transaction state machine and delivers an AR-Confirmed Send confirmation(-) primitive to the requesting FAL ASE.

8.7.3.4 AR-Establish service

8.7.3.4.1 Service overview

This confirmed service operates in a pair-wise manner between two AR endpoints to synchronize their contexts and activate them for the transfer of APDUs.

The endpoint context may be created during or prior to establishment through System Management. Once defined, the Set Attributes and Get Attributes services can be used to update and further coordinate endpoint contexts.

8.7.3.4.2 Service primitives

The service parameters for each primitive are shown in Table 28

Table 28 – AR-Establish service

Parameter name	Req	Cnf
Argument	M	
AREP	M	
Remote DL-Address	M	
Result		M(=)
AREP		M(=)
Error Info		C

AREP
This parameter contains sufficient information to locally identify the AREP to be established.

Remote DL-address
This parameter identifies the remote DL-address.

Result
This parameter indicates whether the service request succeeded or failed.

8.7.3.4.3 Service procedure

8.7.3.4.3.1 AR establishment

The AR-Establish service causes local AREPs to be established independently.

Upon receipt of an AR-Establish request service primitive, the calling FAL issues an AR-Establish confirmation service primitive to the calling AP indicating whether or not it was able to establish the AREP. If one of the following cases is found, it was not able to establish the AREP:

a) the requested AREP is not specified within the FAL, or

b) resources are not available to establish the requested AREP, or

c) for AREPs supported by a data link connection, the Data Link Layer was not able to establish the data link connection.

8.7.3.5 AR-Abort service

8.7.3.5.1 Service overview

The service is used by the FAL User to terminate an AR abruptly. It is always successful; the receiver of an Abort request always aborts the AR. This service may be used on any open AREP, regardless of whether the AR was pre-established or established dynamically using the establish service.

The AR-Abort service is used to instruct the AR ASE to abruptly terminate all activity on an AREP and place it in the closed state. Receipt of an AR-Abort request primitive causes the AR ASE to close the AREP context immediately. The immediate close of each endpoint context causes all outstanding service requests to be cleared. All subsequent service primitives and APDUs received by the AR ASE for the aborted AR are discarded except those of the AR-Establish service.

The AR ASE may also initiate the Abort service if it detects unrecoverable communication failures. In this case, the AR ASE delivers an AR-Abort indication primitive informing the user of the failure and closes the endpoint context.

8.7.3.5.2 Service primitives

The service parameters for each primitive are shown in Table 29.

Table 29 – AR-Abort

Parameter name	Req	Ind
Argument	M	M(=)
AREP	M	M(=)
Originator	M	M(=)
Reason Code	M	M(=)
Additional Detail	U	U(=)

Originator
This parameter identifies the originator of the AR-Abort. Possible values are "DLL", "FAL", or "FAL-USER". The value DLL cannot be used in the request primitive.

Reason Code
This parameter indicates the reason for the Abort. It may be supplied by either the provider or the user. One reason is defined: AR ASE error. Other reason code values may be supplied by the Data Link Layer or by the user.

Additional Detail
This optional parameter contains user data that accompanies the indication. When used, the value submitted in the request primitive is delivered unchanged in the indication primitive.

8.7.3.5.3 Service procedure

The Abort service is a service that operates through a queue.

If the user wishes to abort an AR, it submits an Abort request primitive to its FAL AR ASE. If an AR ASE detects an unrecoverable local failure or a communication failure, it delivers an abort indication primitive to the endpoint user.

The FAL AR ASE also transitions the AREP state to CLOSED.

9 ARs

9.1 General

Application Relationships provided are:

- Point-to-point user-Triggered Confirmed client/server AREP (PTC-AR)
- Point-to-point user-Triggered Unconfirmed client/server AREP (PTU-AR)
- Point-to-point network-Scheduled Unconfirmed publisher/subscriber AREP (PSU-AR)
- Multipoint user-Triggered Unconfirmed publisher/subscriber AREP (MTU-AR)
- Multipoint network-Scheduled Unconfirmed publisher/subscriber AREP (MSU-AR)

9.2 Point-to-point user-Triggered Confirmed client/server AREP (PTC-AR)

9.2.1 Overview

This class is defined to support the on-demand exchange of confirmed services between two application processes. Unconfirmed services are not supported by this type of AR. It uses connectionless-mode Data Link Services for the exchanges. The Data Link Layer transfers may be either acknowledged or unacknowledged. The behaviour of this class is described as

follows. An AR ASE user wishing to convey a request APDU submits it as an AR ASE Service Data Unit to its AREP and the AREP sending the request APDU queues it to its underlying layer for transfer at the next available opportunity.

The AREP receiving the request APDU from its underlying layer queues it for delivery to its AR ASE user in the order in which it was received.

The AREP receiving the request APDU accepts the corresponding response APDU from its AR ASE user and queues it to the underlying layer for transfer.

The AREP that issued the request APDU receives the response APDU from its underlying layer and queues it for delivery to its AR ASE user in the order in which it was received. It also stops its associated service response timer.

The characteristics of this AREP class are summarized as follows:

Roles	Client
	Server
	Peer
Cardinality	one-to-one
Conveyance paths	Bi-directional
Trigger policy	User-triggered
Conveyance policy	Queued

9.2.2 Formal model

FAL ASE:			**AR ASE**
CLASS:			**PTC-AR**
CLASS ID:			**not used**
PARENT CLASS:			**AR ENDPOINT**
ATTRIBUTES:			
1	(m)	Attribute:	Role (CLIENT, SERVER, PEER)
2	(m)	Attribute:	AREP State
3	(m)	Attribute:	Maximum Outstanding Requests Calling
4	(m)	Attribute:	Maximum Outstanding Requests Called
5	(m)	Attribute:	Transmit DL Mapping Reference
6	(m)	Attribute:	Receive DL Mapping Reference
SERVICES:			
1	(m)	OpsService:	Confirmed Send

9.2.3 Attributes

Role
This attribute specifies the role of the AREP. Possible values are:

PEER (CLIENT/SERVER)	Endpoints of this type may perform as either clients or servers, or as both simultaneously.
CLIENT	Endpoints of this type issue confirmed service Request-APDUs to servers and receive confirmed service Response-APDUs.
SERVER	Endpoints of this type receive confirmed service Request-APDUs from clients and issue confirmed service Response-APDUs to them.

AREP State
This attribute specifies the state of the AREP. The values for this attribute are specified in part 3 of this document.

Max Outstanding Requests Calling
This conditional attribute indicates the maximum number of responses that the AREP may be expecting from the remote AREP.

Max Outstanding Requests Called
This conditional attribute indicates the maximum number of responses that the AR endpoint may be expecting from its local user.

Transmit DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the transmit conveyance path for this AREP. DL mappings for the Data Link Layer are specified in part 3 of this document.

Receive DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the receive conveyance path for this AREP. DL mappings for the Data Link Layer are specified in part 3 of this document.

9.2.4 Services

Confirmed Send
This optional service is used to send a confirmed service on an AR.

9.3 Point-to-point user-Triggered Unconfirmed client/server AREP (PTU-AR)

9.3.1 Overview

This class is defined to support the on-demand exchange of unconfirmed services between two application processes. It uses connectionless-mode Data Link Services for the exchanges. The Data Link Layer transfers may be either acknowledged or unacknowledged. The behaviour of this class is described as follows.

An AR ASE user wishing to convey a request APDU submits it as an AR ASE Service Data Unit to its AREP. The AREP sending the request APDU queues it to its underlying layer for transfer at the next available opportunity.

The AREP receiving the request APDU from its underlying layer queues it for delivery to its AR ASE user in the order in which it was received.

The characteristics of this AREP class are summarized as follows:

Roles	Client
	Server
	Peer
Cardinality	one-to-one
Conveyance paths	Unidirectional
Trigger policy	User-triggered
Conveyance policy	Queued

9.3.2 Formal model

FAL ASE:			**AR ASE**
CLASS:			**Point-to-point user-Triggered Unconfirmed client/server AREP**
CLASS ID:			**not used**
PARENT CLASS:			**AR ENDPOINT**
ATTRIBUTES:			
1	(m)	Attribute:	Role (CLIENT, SERVER, PEER)
2	(m)	Attribute:	AREP State
3	(m)	Attribute:	Transmit DL Mapping Reference
4	(m)	Attribute:	Receive DL Mapping Reference
SERVICES:			
1	(m)	OpsService:	Unconfirmed Send

9.3.3 Attributes

Role

This attribute specifies the role of the AREP. Possible values are:

PEER (CLIENT/SERVER)	Endpoints of this type may perform as either clients or servers, or as both simultaneously. Endpoints of this type shall indicate whether they are to be the initiator of the AR establishment process.
CLIENT	Endpoints of this type issue unconfirmed service Request-APDUs to servers.
SERVER	Endpoints of this type receive confirmed and unconfirmed service Request-APDUs from clients.

AREP State
This attribute specifies the state of the AREP. The values for this attribute are specified in Section 3 of this PAS.

Transmit DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the transmit conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

Receive DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the receive conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

9.3.4 Services

Unconfirmed Send
This optional service is used to send an unconfirmed service on an AR.

9.4 Point-to-point network-Scheduled Unconfirmed publisher/subscriber AREP (PSU-AR)

9.4.1 Overview

This class is defined to support the “push” model for scheduled and buffered distribution of unconfirmed services to one application process.

The behaviour of this type of AR can be described as follows.

An AR ASE user wishing to convey a request APDU submits it as an AR ASE Service Data Unit to its AREP for distribution. Sending AREP writes the APDU into the internal buffer, completely replacing the existing contents of the buffer. The AR ASE transfers the buffer contents at the next scheduled transfer opportunity.

If the AREP receives another APDU before the buffer contents are transmitted, the buffer contents will be replaced with the new APDU, and the previous APDU will be lost. When the buffer contents are transmitted, the AR ASE notifies the user of transmission.

At the receiving endpoint, the APDU is received from the network and is written immediately into the buffer, completely overwriting the existing contents of the buffer. The endpoint notifies the user that the APDU has arrived and delivers it to the user according to the local user interface. If the APDU has not been delivered before the next APDU arrives, it will be overwritten by the next APDU and lost.

An FAL user receiving the buffered transmission may request to receive the currently buffered APDU later.

The characteristics of this AREP class are summarized as follows:

Roles	Publisher
	Subscriber
Cardinality	one-to-one
Conveyance paths	Unidirectional
Trigger policy	Network-scheduled
Conveyance policy	Buffered

9.4.2 Formal model

FAL ASE: **AR ASE**
CLASS: **Point-to-point network-Scheduled Unconfirmed publisher/subscriber AREP**
CLASS ID: **not used**
PARENT CLASS: **AR ENDPOINT**
ATTRIBUTES:

1	(m)	Attribute:	Role (PUBLISHER, SUBSCRIBER)
2	(m)	Attribute:	AREP State
3	(m)	Attribute:	Transmit DL Mapping Reference
4	(m)	Attribute:	Receive DL Mapping Reference

SERVICES:

1	(o)	OpsService:	Unconfirmed Send

9.4.3 Attributes

Role
This attribute specifies the role of the AREP. Possible values are:

PUBLISHER Endpoints of this type issue unconfirmed service Request-APDUs to subscribers.

SUBSCRIBER Endpoints of this type receive unconfirmed service Request-APDUs issued by the publisher.

AREP State
This attribute specifies the state of the AREP. The values for this attribute are specified in Section 3 of this PAS.

Transmit DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the transmit conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

Receive DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the receive conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PASt.

9.4.4 Services

Unconfirmed Send
This optional service is used to send an unconfirmed service on an AR.

9.5 Multipoint user-Triggered Unconfirmed publisher/subscriber AREP (MTU-AR)

9.5.1 Overview

This class is defined to support the on-demand exchange of unconfirmed services between two or more application processes. This class uses connectionless Data Link Services for the exchanges. The behaviour of this class is as follows.

An AR ASE user wishing to convey a request APDU submits it as an AR ASE Service Data Unit to its AREP. The AREP sending the request APDU queues it to its underlying layer for transfer at the next available opportunity.

The AREP receiving the request APDU from its underlying layer queues it for delivery to its AR ASE user in the order in which it was received.

The characteristics of this AREP class are summarized as follows:

Roles	Publisher
	Subscriber
Cardinality	one-to-many
Conveyance paths	Unidirectional
Trigger policy	User-triggered
Conveyance policy	Queued

9.5.2 Formal model

FAL ASE:			**AR ASE**
CLASS:			**Multipoint user-Triggered Unconfirmed publisher/subscriber AREP**
CLASS ID:			**not used**
PARENT CLASS:			**AR ENDPOINT**
ATTRIBUTES:			
1	(m)	Attribute:	Role (PUBLISHER, SUBSCRIBER)
2	(m)	Attribute:	AREP State
3	(m)	Attribute:	Transmit DL Mapping Reference
4	(m)	Attribute:	Receive DL Mapping Reference
SERVICES:			
1	(m)	OpsService:	Unconfirmed Send

9.5.3 Attributes

Role
This attribute specifies the role of the AREP. Possible values are:

PUBLISHER	Endpoints of this type issue unconfirmed service Request-APDUs to subscribers.
SUBSCRIBER	Endpoints of this type receive unconfirmed service Request-APDUs issued by the publisher.

AREP State
This attribute specifies the state of the AREP. The values for this attribute are specified in Section 3 of this PAS.

Transmit DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the transmit conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

Receive DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the receive conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

9.5.4 Services

Unconfirmed Send
This optional service is used to send an unconfirmed service on an AR.

9.6 Multipoint network-Scheduled Unconfirmed publisher/subscriber AREP (MSU-AR)

9.6.1 Overview

This class is defined to support the "push" model for scheduled and buffered distribution of unconfirmed services to one or more application processes.

The behaviour of this type of AR can be described as follows.

An AR ASE user wishing to convey a request APDU submits it as an AR ASE Service Data Unit to its AREP for distribution. The sending AREP writes the APDU into the internal buffer, completely replacing the existing contents of the buffer.

The AREP transfers the buffer contents at the next scheduled transfer opportunity.

If the AREP receives another APDU before the buffer contents are transmitted, the buffer contents will be replaced with the new APDU, and the previous APDU will be lost. When the buffer contents are transmitted, the AR ASE notifies the user of transmission.

At the receiving endpoint, the APDU is received from the network and is written immediately into the buffer, completely overwriting the existing contents of the buffer. The endpoint notifies the user that the APDU has arrived and delivers it to the user according to the local user interface. If the APDU has not been delivered before the next APDU arrives, it will be overwritten by the next APDU and lost.

An FAL user receiving the buffered transmission may request to receive the currently buffered APDU later.

The characteristics of this AREP class are summarized as follows:

Roles	Publisher
	Subscriber
Cardinality	one-to-many
Conveyance paths	Unidirectional
Trigger policy	Network-scheduled
Conveyance policy	Buffered

9.6.2 Formal model

FAL ASE:			**AR ASE**
CLASS:			**Multipoint network-Scheduled Unconfirmed publisher/subscriber AREP**
CLASS ID:			**not used**
PARENT CLASS:			**AR ENDPOINT**
ATTRIBUTES:			
1	(m)	Attribute:	Role (PUBLISHER, SUBSCRIBER)
2	(m)	Attribute:	AREP State
3	(m)	Attribute:	Transmit DL Mapping Reference
4	(m)	Attribute:	Receive DL Mapping Reference
SERVICES:			
1	(o)	OpsService:	Unconfirmed Send

9.6.3 Attributes

Role
This attribute specifies the role of the AREP. Possible values are:

PUBLISHER Endpoints of this type publish their data issuing unconfirmed service Request-APDUs.

SUBSCRIBER Endpoints of this type receive unconfirmed service Request-APDUs issued by publishers.

AREP State
This attribute specifies the state of the AREP. The values for this attribute are specified in Section 3 of this PAS.

Transmit DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the transmit conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

Receive DL Mapping Reference
This attribute provides a reference to the underlying Data Link Layer mapping for the receive conveyance path for this AREP. DL mappings for the Data Link Layer are specified in Section 3 of this PAS.

9.6.4 Services

Unconfirmed Send
This optional service is used to send an unconfirmed service on an AR.

10 Summary of FAL classes

This subclause contains a summary of the defined FAL Classes. The Class ID values have been assigned to be compatible with existing standards.

Table 30 provides a summary of the classes.

Table 30 – FAL class summary

FAL ASE	Class
Variable	**Variable**
	Variable List
Event	**Notifier**
Load region	**Load region**
Function invocation	**Function invocation**
Time	**Time**
Network Management	**Network Manager**
Application relationship	**AREP**
	PTC-AREP
	PTU-AREP
	PSU-AREP
	MTU-AREP
	MSU-AREP

11 Permitted FAL services by AREP role

Table 31 below defines the valid combinations of services and AR types (which service APDUs can be sent or received by AR with the specified type). "Unc" and "Cnf" columns indicate whether the service listed in the left-hand column is unconfirmed or confirmed respectively.

Table 31 – FAL services by AR type

	Unc	Cnf	Client		Server		Publisher		Subscriber	
FAL Services			req	rcv	req	rcv	req	rcv	req	rcv
Variable ASE										
Read		X	X			X				
Write		X	X			X				
Information Report	X				X		X			X
Event ASE										
Event Notification	X						X			X
Notification Recovery	X							X	X	
Load Region ASE										
Download		X	X			X				
Upload		X	X			X				
Function invocation ASE										
Start		X	X			X				
Stop		X	X			X				
Resume		X	X			X				
Time ASE										
Get Network Time		X								
Set Network Time	X						X			X
Tick Notification service	X						X			X
NW Management ASE										
Get Network Status		X								
Get Station Status		X								
NW Status Report	X						X			X
Station Status Report	X						X			X
AR ASE										
PTC-AR		X	X			X				
PTU-AR	X		X			X				
PSU-AR	X			X	X					
MTU-AR	X						X			X
MSU-AR	X						X			X

Section 3: Application Layer protocol specification

12 Abstract syntax

12.1 FAL PDU abstract syntax

12.1.1 Top level definition

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

12.1.2 FalArHeader

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

12.1.3 Confirmed send service

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

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

12.1.4 Unconfirmed send service

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

12.2 Abstract syntax of PDU Body

12.2.1 ConfirmedServiceRequest PDUs

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

12.2.2 ConfirmedServiceResponse PDUs

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

12.2.3 Unconfirmed PDUs

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

12.2.4 Error information

12.2.4.1 Error type

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

12.2.4.2 Error class

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

12.3 PDUs for ASEs

12.3.1 PDUs for Variable ASE

12.3.1.1 Read service PDUs

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

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

12.3.1.2 Write service PDUs

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


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

12.3.1.3 Information Report service PDUs

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

12.3.2 PDUs for Event ASE

12.3.2.1 Event Notification service

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

12.3.2.2 Notification Recovery service

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

12.3.3 PDUs for Load region ASE

12.3.3.1 Download service

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


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

12.3.3.2 Upload service

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


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

12.3.4 PDUs for Function Invocation ASE

12.3.4.1 Start service

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


Start-ResponsePDU ::= ErrorType
```

12.3.4.2 Stop service

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


Stop-ResponsePDU ::= ErrorType
```

12.3.4.3 Resume services

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


Resume-ResponsePDU ::= ErrorType
```

12.3.5 PDUs for Time ASE

12.3.5.1 Time service

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

Time-ResponsePDU ::= Time-PDU

TimeDistribute-RequestPDU ::= Time-PDU

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

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

12.3.6 PDUs for Network Management ASE

12.3.6.1 Network Management service

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

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

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

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

12.4 Type definitions

12.4.1 Variable ASE types

There are no types special for the Variable ASE.

12.4.2 Event ASE types

```
Ev_SequenceNumber ::= Unsigned8

Ev_EventData ::= ANY

En_EventCount ::= Unsigned8

Ev_TimeTag ::= Unsigned16
```

12.4.3 Load Region ASE types

There are no types special for the Load Region ASE.

12.4.4 Function Invocation ASE types

There are no types special for the function Invocation ASE.

12.4.5 Time ASE types

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

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

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


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

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

12.4.6 Network Management ASE types

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


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

```
Nm_PublicKey ::= Unsigned64

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

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

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

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

12.4.7 General types

12.4.7.1 Gn_KeyAttribute

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

12.4.7.2 Gn_Name

```
Gn_Name ::= OctetString
```

12.4.7.3 Gn_NumericAddress

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

12.4.7.4 Gn_NumericID

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

12.4.7.5 Gn_SymbolicAddress

```
Gn_SymbolicAddress ::= VisibleString
```

12.4.7.6 Gn_FullyNestedTypeDescription

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

12.5 Data types

12.5.1 Notation for the Boolean type

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

12.5.2 Notation for the Integer type

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

12.5.3 Notation for the Unsigned type

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

12.5.4 Notation for the Floating Point type

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

12.5.5 Notation for the BitString type

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

12.5.6 Notation for the OctetString type

```
OctetString ::= OCTET STRING                -- For generic use
OctetString2 ::= OCTET STRING SIZE (2)      -- Fixed two-octet octet string
OctetString4 ::= OCTET STRING SIZE (4)      -- Fixed four-octet octet string
OctetString6 ::= OCTET STRING SIZE (6)      -- Fixed six-octet octet string
OctetString7 ::= OCTET STRING SIZE (7)      -- Fixed seven-octet octet string
OctetString8 ::= OCTET STRING SIZE (8)      -- Fixed eight-octet octet string
OctetString16 ::= OCTET STRING SIZE (16)    -- Fixed 16 octet octet string
```

12.5.7 Notation for VisibleString type

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

12.5.8 Notation for the UNICODEString type

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

12.5.9 Notation for Binary Time type

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

12.5.10 Notation for BCD type

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

12.5.11 Notation for Compact Boolean Array type

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

12.5.12 Notation for Compact BCD Array type

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

13 Transfer syntax

13.1 Overview of encoding

The FAL-PDUs encoded shall have a uniform format. The FAL-PDUs shall consist of two major parts, the “APDU Header” part and the “APDU Body” part as shown in Figure 3.

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

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

Figure 3 – APDU overview

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

NOTE The Data Link Layer service provides a DLSDU parameter that implies the length of the APDU. Thus, the APDU length information is not included in the APDU.

13.2 APDU Header encoding

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

They are shown in Figure 3.

13.2.1 Encoding of FalArHeader Field

All the FAL PDUs shall have the common PDU-header called FalArHeader. The FalArHeader identifies abstract syntax, transfer syntax, and each of the PDUs. Table 32 defines how this header shall be used.

Table 32 – Encoding of FalArHeader Field

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

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

13.2.2 Encoding of Type Field

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

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

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

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

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

c) Figure 4 illustrates the encoding of the Type Field.

Figure 4 – Type Field

13.2.3 Encoding of InvokeID Field

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

13.3 APDU Body encoding

13.3.1 General

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

Identifier Octet
Length Octet(s)
Contents Octet(s)

13.3.2 Identifier Octet

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

It consists of the P/C flag and the Tag field as shown in Figure 5.

Figure 5 – Identifier Octet

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

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

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

13.3.3 Length Octet(s)

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

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

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

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

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

Figure 6 and Figure 7 depict encoding examples of the Length Octet(s):

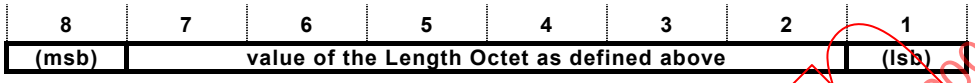

Figure 6 – Length octet (one-octet format)

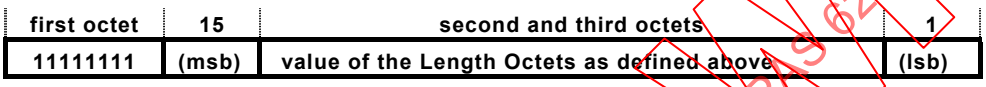

Figure 7 – Length octets (three-octet format)

13.3.4 Contents Octet(s)

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

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

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

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

13.4 Data type encoding rules

13.4.1 General

13.4.1.1 Boolean

A Boolean value shall be encoded as follows:

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

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

13.4.1.2 Integer

An Integer value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

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

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

13.4.1.3 Unsigned Value

An Unsigned Value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

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

13.4.1.4 Floating Point

A Floating Point value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

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

13.4.1.5 Bit String

A Bit String value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

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

13.4.1.6 Octet String

An Octet String value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

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

13.4.1.7 Visible String

A Visible String value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

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

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

13.4.1.8 UNICODE String (ISO 10646 String)

A UNICODE String value shall be encoded as follows:

a) The Identifier Octet shall not be present.

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

c) Each ISO10646 character shall be placed in two octets in the Contents Octet(s), with the high-order octet placed in the first octet and the low-order octet in the subsequent octet, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octet(s).

13.4.1.9 Binary Time

A Binary Time value shall be encoded as follows:

a) The Identifier Octet shall not be present

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

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

13.4.1.10 BCD type

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

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

13.4.1.11 Compact Boolean Array

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

13.4.1.12 Compact BCD Array type

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

b) The Identifier Octet shall not be present.

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

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

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

13.4.1.13 SEQUENCE type

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

a) The Identifier Octet shall not be present.

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

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

13.4.1.14 SEQUENCE OF type

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

a) The Identifier Octet shall not be present.

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

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

13.4.1.15 CHOICE type

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

a) The Identifier Octet shall not be present.

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

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

13.4.1.16 Null

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

a) The Identifier Octet shall not be present.

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

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

13.4.1.17 Tagged type

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

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

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

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

13.4.1.18 IMPLICIT type

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

a) The Identifier Octet shall not be present.

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

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

13.4.1.19 OPTIONAL and DEFAULT types

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

a) The Identifier Octet shall not be present.

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

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

13.4.1.20 ANY type

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

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

a) The Identifier Octet shall not be present.

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

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

14 FAL protocol state machines structure

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

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

The behaviour of the FAL is described by three integrated protocol machines. The three kinds of protocol machines are: FAL Service Protocol Machines (FSPMs), the Application Relationship Protocol Machines (ARPMs), and the Data Link Layer Mapping Protocol Machines (DMPMs). Specific protocol machines are defined for different AREP types. The relationships among these protocol machines as well as primitives exchanged among them are depicted in Figure 8.

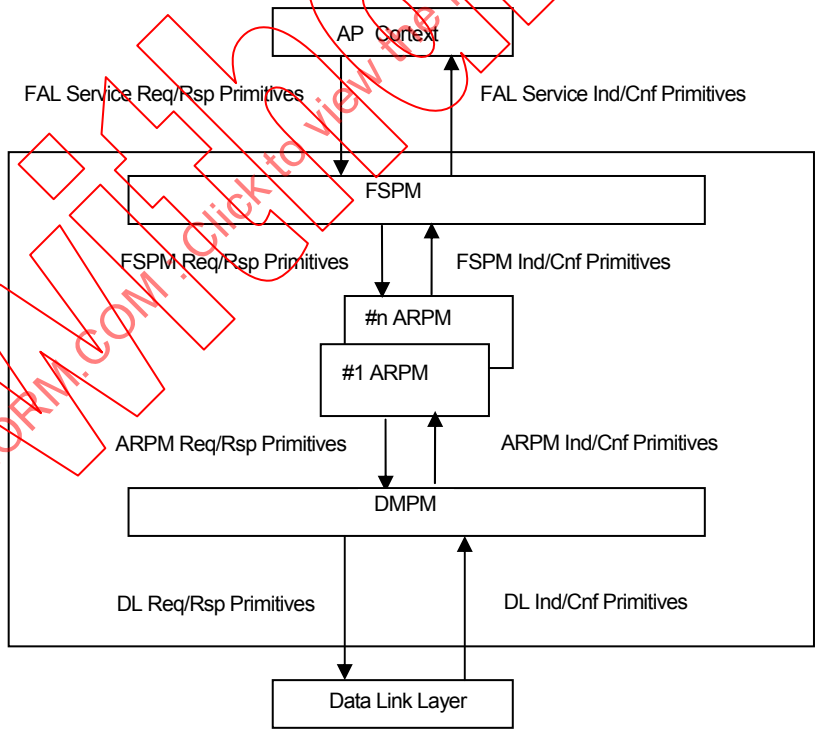

Figure 8 – Relationships among protocol machines and adjacent layers

The FSPM is responsible for the following activities:

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

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

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

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

The ARPM is responsible for the following activities:

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

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

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

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

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

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

15 AP-Context state machine

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

16 FAL Service Protocol Machines (FSPMs)

16.1 General

There are FAL Service Protocol Machines as follows:

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

16.2 Common parameters of the primitives

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

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

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

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

16.3 Variable ASE Protocol Machine (VARM)

16.3.1 Primitive definitions

16.3.1.1 Primitives exchanged

Table 33 shows the service primitives, including their associated parameters exchanged between the FAL user and the VARM.

Table 33 – Primitives exchanged between FAL user and VARM

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

16.3.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the VARM are listed in Table 34.

Table 34 – Parameters used with primitives exchanged FAL user and VARM

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

16.3.2 State machine

16.3.2.1 General

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

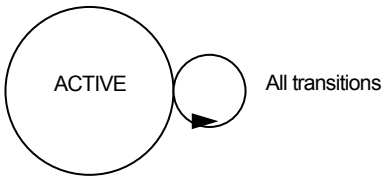

Figure 9 – State transition diagram of VARM

16.3.2.2 State tables

The VARM state machine is described in Figure 9, and in Table 35 and Table 36.

Table 35 – VARM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

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

Table 36 – VARM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

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

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

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

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

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

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

16.3.2.3 Functions

Table 37 lists the functions used by the VARM, their arguments and their descriptions.

Table 37 – Functions used by the VARM

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

16.4 Event ASE Protocol Machine (EVTM)

16.4.1 Primitive definitions

16.4.1.1 Primitives exchanged

Table 38 shows the service primitives, including their associated parameters exchanged between the FAL user and the EVTM.

Table 38 – Primitives exchanged between FAL user and EVTM

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

16.4.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the EVTM are listed in Table 39.

Table 39 – Parameters used with primitives exchanged FAL user and EVTM

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

16.4.2 State machine

16.4.2.1 General

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

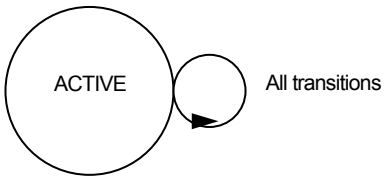

Figure 10 – State transition diagram of EVTM

16.4.2.2 State tables

The EVTM state machine is described in Figure 10, and in Table 40 and Table 41.

Table 40 – EVTM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

Table 41 – EVTM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

16.4.2.3 Functions

Table 42 lists the function used by the EVTM, their arguments, and their description.

Table 42 – Functions used by the EVTM

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

16.5 Load Region ASE Protocol Machine (LDRM)

16.5.1 Primitive definitions

16.5.1.1 Primitives exchanged

Table 43 shows the service primitives, including their associated parameters exchanged between the FAL user and the LDRM.

Table 43 – Primitives exchanged between FAL user and LDRM

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

16.5.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the LDRM are listed in Table 44.

Table 44 – Parameters used with primitives exchanged FAL user and LDRM

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

16.5.2 State machine

16.5.2.1 General

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

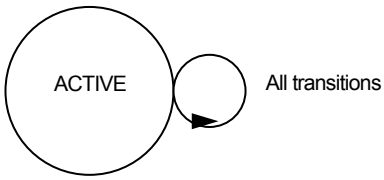

Figure 11 – State transition diagram of LDRM

16.5.2.2 State tables

The LDRM state machine is described in Figure 11, and in Table 45 and Table 46.

Table 45 – LDRM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 46 – LDRM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

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

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

16.5.2.3 Functions

Table 47 lists the functions used by the LDRM, their arguments, and their descriptions.

Table 47 – Functions used by the LDRM

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

16.6 Function Invocation ASE Protocol Machine (FNIM)

16.6.1 Primitive definitions

16.6.1.1 Primitives exchanged

Table 48 shows the service primitives, including their associated parameters exchanged between the FAL user and the FNIM.

Table 48 – Primitives exchanged between FAL user and FNIM

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

16.6.1.2 Parameters of primitives

The parameter used with the primitives exchanged between the FAL user and the FNIM is listed in Table 49.

Table 49 – Parameters used with primitives exchanged FAL user and FNIM

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

16.6.2 State machine

16.6.2.1 General

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

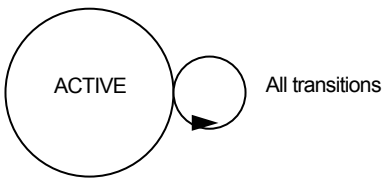

Figure 12 – State transition diagram of FNIM

16.6.2.2 State tables

The FNIM state machine is described in Figure 12, and in Table 50 and Table 51.

Table 50 – FNIM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

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

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

Table 51 – FNIM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

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

#	Current state	Event or condition => action	Next state

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

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

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

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

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

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

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

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

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

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

16.6.2.3 Functions

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

Table 52 – Functions used by the FNIM

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

16.7 Time ASE Protocol Machine (TIMM)

16.7.1 Primitive definitions

16.7.1.1 Primitives exchanged

Table 53 shows the service primitives, including their associated parameters exchanged between the FAL user and the TIMM.

Table 53 – Primitives exchanged between FAL user and TIMM

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

16.7.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the TIMM are listed in Table 54.

Table 54 – Parameters used with primitives exchanged FAL user and TIMM

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

16.7.2 State machine

16.7.2.1 General

The TIMM State Machine has four possible states. The defined states and their descriptions are shown in Table 55 and Figure 13.

Table 55 – TIMM states

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

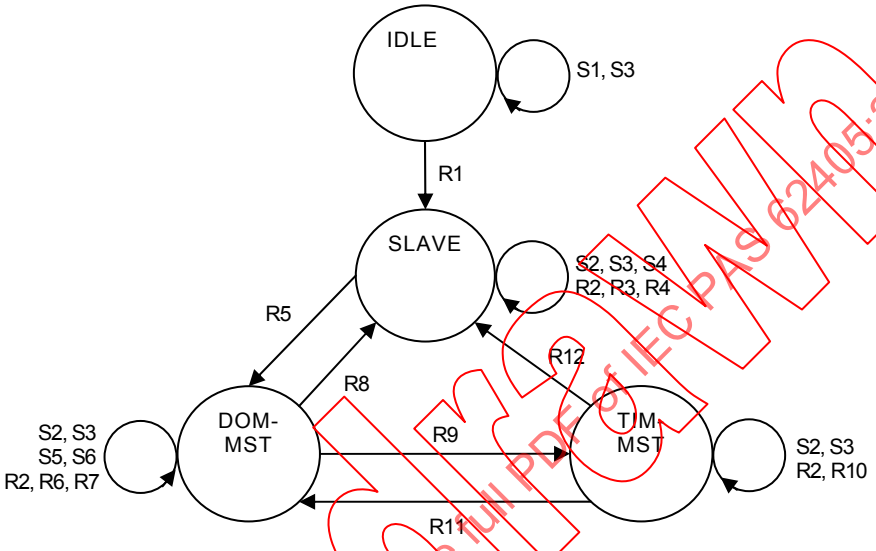

Figure 13 – State transition diagram of TIMM

16.7.2.2 State tables

The TIMM state machine is described in Figure 13, and in Table 56 and Table 57.

Table 56 – TIMM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

S4	SLAVE	CheckTimer(Timer1) = “Expired” => SelectArep (“DOM-MST”, “PTC-AR”), CS_req { user_data := Time-RequestPDU }, StartTimer(Timer1)	SLAVE

#	Current state	Event or condition => action	Next state
S5	DOM-MST	CheckTimer(Timer2) = “Expired” => SelectArep(“DOM”, “MTU-AR”), UCS_req { user_data := TimeDistribute-RequestPDU }, StartTimer(Timer2)	DOM-MST
S6	DOM-MST	CheckTimer(Timer3) = “Expired” => SelectArep(“TIM-MST”, “PTC-AR”), CS_req { user_data := Time-RequestPDU }, StartTimer(Timer3)	DOM-MST

Table 57 – TIMM state table – Receiver transitions

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

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

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

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

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

16.7.2.3 Functions

Table 58 lists the functions used by the TIMM, their arguments, and their descriptions.

Table 58 – Functions used by the TIMM

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

16.8 Network Management ASE Protocol Machine (NWMM)

16.8.1 Primitive definitions

16.8.1.1 Primitives exchanged

Table 59 shows the service primitives, including their associated parameters exchanged between the FAL user and the NWMM.

Table 59 – Primitives exchanged between FAL user and NWMM

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

16.8.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL user and the NWMM are listed in Table 60.

Table 60 – Parameters used with primitives exchanged FAL user and NWMM

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

16.8.2 State machine

16.8.2.1 General

The NWMM State Machine has three possible states. The defined states and their descriptions are shown in Table 61 and Figure 14.

Table 61 – NWMM states

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

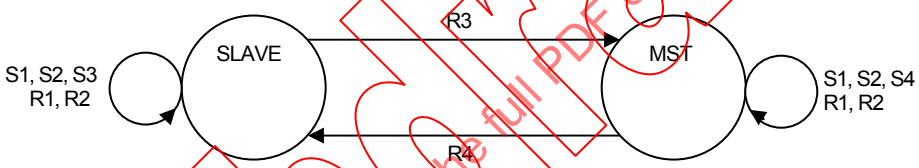

Figure 14 – State transition diagram of NWMM

16.8.2.2 State tables

The NWMM state machine is described in Figure 14, and in Table 62 and Table 63.

Table 62 – NWMM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 63 – NWMM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

16.8.2.3 Functions

Table 64 lists the functions used by the NWMM, their arguments, and their descriptions.

Table 64 – Functions used by the NWMM

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

17 Application Relationship Protocol Machines (ARPMs)

17.1 General

This fieldbus has Application Relationship Protocol Machines (ARPMs) for:

- Point-to-point user-Triggered Confirmed client/server AREP (PTC-AR)
- Point-to-point user-Triggered Unconfirmed client/server AREP (PTU-AR)
- Point-to-point network-Scheduled Unconfirmed client/server AREP (PSU-AR)

- Multipoint user-Triggered Unconfirmed publisher/subscriber AREP (MTU-AR)
- Multipoint network-Scheduled Unconfirmed publisher/subscriber AREP (MSU-AR)

17.2 Primitive definitions

17.2.1 Primitives exchanged

Table 65 lists the primitives, including their associated parameters exchanged between the FSPM and the ARPM.

Table 65 – Primitives exchanged between FSPM and ARPM

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

17.2.2 Parameters of primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are listed in Table 66.

Table 66 – Parameters used with primitives exchanged FSPM user and ARPM

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

17.3 State machine

17.3.1 Point-to-point user-Triggered Confirmed client/server ARPM (PTC-ARPM)

17.3.1.1 General

The PTC-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 67 and Figure 15.

Table 67 – PTC-ARPM states

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

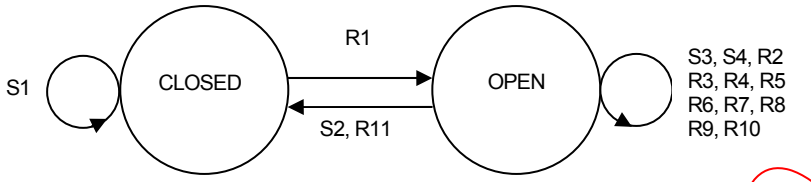

Figure 15 – State transition diagram of the PTC-ARPM

17.3.1.2 States

The PTC-ARPM state machine is described in Figure 15, and in Table 68 and Table 69.

Table 68 – PTC-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 69 – PTC-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

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

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

17.3.2 Point-to-point user-Triggered Unconfirmed client/server ARPM (PTU-ARPM)

17.3.2.1 General

The PTU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 70 and Figure 16.

Table 70 – PTU-ARPM states

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

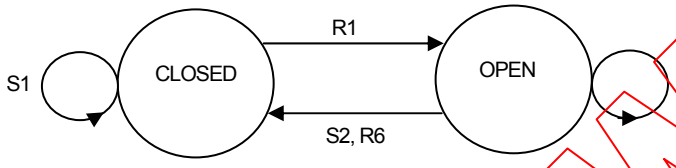

Figure 16 – State transition diagram of the PTU-ARPM

17.3.2.2 State tables

The PTU-ARPM state machine is described in Figure 16, and in Table 71 and Table 72.

Table 71 – PTU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 72 – PTU-ARPM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

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

17.3.3 Point-to-point network-Scheduled Unconfirmed client/server ARMP (PSU-ARPM)

17.3.3.1 General

The PSU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 73 and Figure 17.

Table 73 – PSU-ARPM states

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

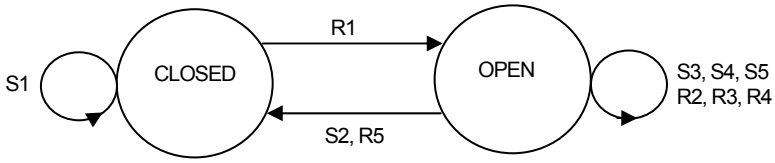

Figure 17 – State transition diagram of the PSU-ARPM

17.3.3.2 State tables

The PSU-ARPM state machine is described in Figure 17, and in Table 74 and Table 75.

Table 74 – PSU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

Table 75 – PSU-ARPM state table – Receiver transitions

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

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

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

17.3.4 Multipoint user-Triggered Unconfirmed publisher/subscriber ARPM (MTU-ARPM)

17.3.4.1 General

The MTU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 76 and Figure 18.

Table 76 – MTU-ARPM states

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

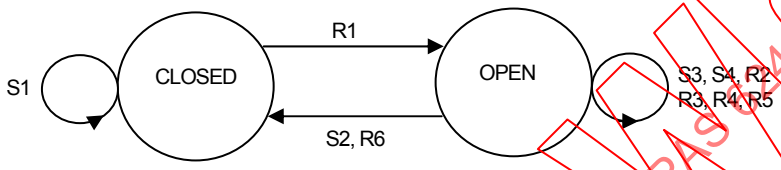

Figure 18 – State transition diagram of the MTU-ARPM

17.3.4.2 State tables

The MTU-ARPM state machine is described in Figure 18, and in Table 77 and Table 78.

Table 77 – MTU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 78 – MTU-ARPM state table – Receiver transitions

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

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

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

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

17.3.5.1 General

The MSU-ARPM State Machine has two possible states. The defined states and their descriptions are shown in Table 79 and Figure 19.

Table 79 – MSU-ARPM states

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

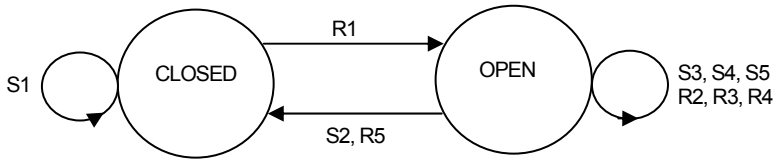

Figure 19 – State transition diagram of the MSU-ARPM

17.3.5.2 State tables

The MSU-ARPM state machine is described in Figure 19, and in Table 80 and Table 81.

Table 80 – MSU-ARPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

Table 81 – MSU-ARPM state table – Receiver transitions

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

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

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

17.4 Functions

Table 82 lists the functions used by the ARPMs, their arguments, and their descriptions.

Table 82 – Functions used by the ARPMs

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

18 DLL Mapping Protocol Machine (DMPM)

18.1 General

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

The primitives issued by ARPM to DMPM are passed to the Data Link Layer as the DLS primitives. The primitives issued to DMPM from the Data Link Layer are notified to an appropriate ARPM out of the ARPMs.

DMPM adds and deletes parameters to/from the primitives exchanged between ARPM and the Data Link Layer if necessary.

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

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

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

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

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

18.2 Primitive definitions

18.2.1 Primitives exchanged between DMPM and ARPM

Table 83 lists the primitives exchanged between the DMPM and the ARPM.

Table 83 – Primitives exchanged between DMPM and ARPM

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

18.2.2 Primitives exchanged between Data Link Layer and ARPM

Table 84 lists the primitives exchanged between the Data Link Layer and the ARPM.

Table 84 – Primitives exchanged between Data Link Layer and DMPM

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

18.2.3 Parameters of DMPM/Data Link Layer primitives

The parameters used with the primitives exchanged between the DMPM and the Data Link Layer are identical to those defined in Section 4 of this PAS. They are prefixed by “dl_” to indicate that they are used by the FAL.

18.3 DMPM state machine

18.3.1 DMPM states

The DMPM State Machine has only one possible state. The defined state and their descriptions are shown in Table 85 and Figure 20.

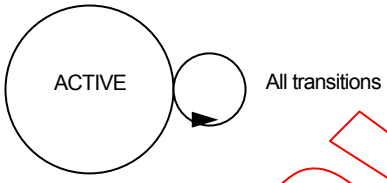

Figure 20 – State transition diagram of DMPM

Table 85 – DMPM states

State	Description
ACTIVE	The DMPM in the ACTIVE state is ready to transmit or receive primitives to or from the Data Link Layer and the ARPM.

18.3.2 DMPM state table

The DMPM state machine is described in Table 86 and Table 87

Table 86 – DMPM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

#	Current state	Event or condition => action	Next state

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

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

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

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

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

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

Table 87 – DMPM state table – Receiver transitions

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

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

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

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

18.3.3 Functions used by DMPM

Table 88 contains the functions used by the DMPM, their arguments and their descriptions.

Table 88 – Functions used by the DMPM

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

Section 4: Data Link Service definition

19 Overview

19.1 General

The Data Link Service (DLS) provides transparent and reliable data transfer between DLS-users. It makes the way that supporting communication resources are utilized invisible to DLS-users.

In particular, the DLS provides the following:

a) Independence from the underlying Physical Layer. The DLS relieves DLS-users from all direct concerns regarding which configuration is available (for example, direct connection, or indirect connection through one or more bridges) and which physical facilities are used (for example, which of a set of diverse physical paths).

b) Transparency of transferred information. The DLS provides the transparent transfer of DLS-user-data. It does not restrict the content, format or coding of the information, nor does it ever need to interpret the structure or meaning of that information. It may, however, restrict the amount of information that can be transferred as an indivisible unit.

c) Reliable data transfer. The DLS relieves the DLS-user from concerns regarding insertion or corruption of data, or, if requested, loss, duplication or misordering of data, which can occur. In some cases of unrecovered errors in the Data Link Layer, duplication or loss of DLSDUs can occur. In cases where protection against misordering of data is not employed, misordering can occur.

d) Quality of Service (QoS) selection. The DLS provides DLS-users with a means to request and to agree upon a quality of service for the data transfer. QoS is specified by means of QoS parameters representing aspects such as mode of operation, transit delay, accuracy, reliability, security and functional safety.

e) Addressing. The DLS allows the DLS-user to identify itself and to specify the DLSAPs to/from which data are to be transferred.

f) Scheduling. The DLS allows the set of DLS-users to provide some guidance on internal scheduling of the distributed DLS-provider. This guidance supports the time-critical aspects of the DLS by permitting the DLS-user some degree of management over when opportunities for communication will be granted to various DLEs for various DLSAP-addresses.

g) Common time sense. The DLS can provide the DLS-user with a sense of time that is common to all DLS-users on the network.

h) Queues. The DLS can provide the sending or receiving DLS-user with a FIFO queue, where each queue item can hold a single DLSDU.

19.2 Overview of network structure

Although the DLS conforms formally to the “three-layer” Fieldbus Reference Model, it actually utilizes the Transport Layer Service and Network Layer Service in addition to the Data Link Layer Service of the OSI Basic Reference Model. The DLS of this specification is actually a Transport Layer Service in terms of the OSI Basic Reference Model. Thus the network may consist of one or more subnetworks interconnected to each other by the Network Layer Relay entities, known as routers.

A network may be redundant structure. A redundant network consists of two independent networks making dual-redundancy; they are referred to as the primary network and secondary network. Consequently, dual-redundant independent logical communication channels between two communication end nodes can be implemented. This logical channel is called a route.

A pair of subnetworks comprising a dual-redundant network is called a domain.

A subnetwork consists of one or more segments interconnected by DL-relay entities, known as bridges. The topology of a subnetwork may be a tree, a ring or a mesh consisting of segments interconnected by bridges.

A segment consists of one or more DLEs, all of which are connected directly (i.e., without intervening DL-relay entities) to a single shared logical communication channel, which is called a link.

A path (logical communication channel) consists of one or two physically independent and logically parallel real communication channels, which are called links.

19.3 Overview of addressing

domain number
a numeric identifier that indicates a domain. Two subnetworks comprising a dual-redundant domain have an identical domain number.

station number
a numeric identifier that indicates a Vnet/IP station. Two end nodes comprising a dual-redundant station have an identical station number.

TSAP address
the DL-entity actually provides Transport Layer Services, so DLS is provided at TSAPs. TSAP is identified by a set of TSAP-address (IP-Address) and TSAP-identifier (UDP Port Number).

IP address
a unique address for each end node. An IP address consists of a network address portion and a host address portion. The network address is assigned according to the domain number, while the host address is assigned based on the station number. Each end node of a dual-redundant station has a different host address.

MAC address
MAC address is a unique address for an end node defined in ISO/IEC 8802-3. The destination MAC address is resolved by the mechanism defined in RFC 826 from the destination IP address.

19.4 Types of Data Link Service

There are three types of DLS:

a) a DLSAP management service

b) a connectionless-mode data transfer service

c) a DL-management service

20 DLSAP management service

20.1 Overview

This clause provides a conceptual definition of the services provided by the DLS-provider to the DLS-user(s). Nothing in this section shall be construed to constrain the actual implementations of the interactions at the DLS-provider to DLS-user interface.

20.2 Facilities of the DLSAP management service

The DLS provides the following facilities to the DLS-user:

a) a means for creating and deleting a FIFO queue of specified depth.

b) a means for assigning a DLSAP-address to the DLSAP.

c) a means for binding previously created FIFO queues to an each potential direction of connectionless data transfer at the specified DLSAP.

d) a means for specifying QoS parameters of the specified DLSAP.

e) a means for releasing resources used previously for the DLSAP.

20.3 Model of the DLSAP management service

This part of the PAS uses the abstract model for a layer service defined in ISO/IEC 10731, Clause 5. The model defines interactions between the DLS-user and the DLS-provider that take place at a DLSAP. Information is passed between the DLS-user and the DLS-provider by DLS primitives that convey parameters.

The DLSAP management primitives are used to provide a local service between a DLS-user and the local DLE. Remote DLEs and remote DLS-users are not involved directly, so there is no need for the other primitives of ISO/IEC 10731. Therefore the DLSAP management services are provided by request primitives with input and output parameters.

20.4 Sequence of primitives at one DLSAP

Table 89 is a summary of the DLSAP management primitives and parameters. The major sequence of primitives at a single DLE is shown in Figure 21.

Table 89 – Summary of DLSAP management primitives and parameters

Service	Primitive	Parameter
queue creation	DL-CREATE request	(*in* Queue DLS-user-identifier, Maximum DLSDU size, Maximum queue depth, *out* Status, Queue DL-identifier)
queue deletion	DL-DELETE request	(*in* Queue DL-identifier, *out* Status)
DLSAP activation	DL-BIND request	(*in* DLSAP-address DLS-user-identifier, Sending queue DL-identifier Receiving queue DL-identifier, QoS parameters, *out* Status, DLSAP-address DL-identifier)
DLSAP deactivation	DL-UNBIND request	(*in* DLSAP-address DL-identifier *out* Status)

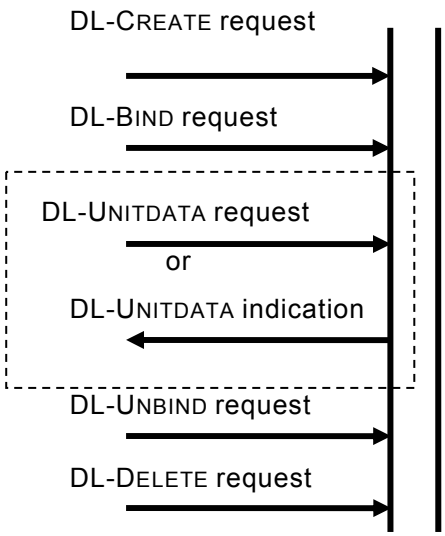

Figure 21 – Sequence of primitives for the DLSAP management DLS

20.5 Create

20.5.1 Function

The create queue DLS primitive may be used to create a limited-depth FIFO queue for later constrained association with a DLSAP. The resulting FIFO queue initially will be empty.

20.5.2 Types of parameter

Table 90 lists the primitive and parameters of the create queue DLS.

Table 90 – DLSAP-management CREATE primitive and parameters

DL-CREATE Parameter name	Request	
	input	output
Queue DLS-user-identifier	M	
Maximum DLSDU size	M	
Maximum queue depth	M	
Status		M
Queue DL-identifier		C

20.5.2.1 Queue DLS-user-identifier

This parameter specifies a means of referring to the queue in output parameters of other local DLS primitives that convey the name of the queue from the local DLE to the local DLS-user.

The naming-domain of the queue DLS-user-identifier is the DLS-user's local-view.

20.5.2.2 Maximum DLSDU size

This parameter specifies an upper bound on the size (in octets) of DLSDUs that can be put into the queue.

20.5.2.3 Maximum queue depth

This parameter specifies the maximum number of items in the associated queue.

20.5.2.4 Status

This parameter allows the DLS-user to determine whether the requested DLS was provided successfully, or failed for the reason specified. The possible value conveyed in this parameter is as follows:

a) “success”;

b) “failure — insufficient resources”;

c) “failure — parameter violates management constraint”;

d) “failure — number of requests violates management constraint”; or

e) “failure — reason unspecified”.

NOTE Addition to, or refinement of, this list of values, to convey more specific diagnostic and management information is permitted.

20.5.2.5 Queue DL-identifier

This parameter is present when the Status parameter indicates that the DL-CREATE request primitive was successful. The queue DL-identifier parameter gives the local DLS-user a means of referring to the queue in input parameters of other local DLS primitives that convey the name of the queue from the local DLS-user to the local DLE.

20.6 Delete

20.6.1 Function

The delete queue DLS primitive may be used to delete a queue created by an earlier create queue DLS primitive.

20.6.2 Types of parameter

Table 91 lists the primitive and parameters of the delete queue DLS.

Table 91 – DLSAP-management DELETE primitive and parameters

DL-DELETE Parameter name	Request	
	input	output
Queue DL-identifier	M	
Status		M

20.6.2.1 Queue DL-identifier

This parameter specifies the local queue to be deleted. Its value shall be the queue DL-identifier returned by a successful prior DL-CREATE request primitive. The DLS-provider will release the local DL-identifier and associated DLS-provider resources.

The DLS-user shall not delete a queue that is still associated with a DLSAP.

20.6.2.2 Status

This parameter allows the DLS-user to determine whether the requested DLS was provided successfully, or failed for the reason specified. The value conveyed in this parameter is as follows:

a) “success”;

b) “failure — resource in use”; or

c) “failure — reason unspecified”.

NOTE Addition to, or refinement of, this list of values, to convey more specific diagnostic and management information is permitted.

20.7 Bind

20.7.1 Function

The bind DLSAP-address DLS primitive is used

a) to associate a DLSAP-address with the DLSAP at which the primitive is invoked;

b) to associate previously created limited depth FIFO queues with the various priorities and directions of potential data transfer at the specified DLSAP-address; and

c) to specify values for some Quality of Service (QoS) attributes for connectionless data transfer services using the specified DLSAP-address.

20.7.2 Types of parameter

Table 92 lists the primitive and parameters of the Bind DLSAP-address DLS.

Table 92 – DLSAP management BIND primitive and parameters

DL-BIND Parameter name	Request input	 output
DLSAP-address DLS-user-identifier	M	
Sending queue DL-identifier	M	
Receiving queue DL-identifier	M	
QoS parameters		
DL Service subtype	M	
DLL maximum confirm delay	U	
DLL priority	U	
Authentication level	U	
Maximum residual error rate	U	
Transmission Timing Window	U	
Status		M
DLSAP-address DL-identifier		C

20.7.2.1 DLSAP-address DLS-user-identifier

This parameter specifies a means of referring to the DLSAP-address in output parameters of other local DLS primitives that convey the name of the DLSAP-address from the local DLE to the local DLS-user.

20.7.2.2 Sending queue DL-identifier

This parameter specifies the local DL-identifier for sending, which has been returned by a successful prior DL-CREATE request primitive that created a queue, which has not yet been deleted.

20.7.2.3 Receiving queue DL-identifier

This parameter specifies the local DL-identifier for receiving, which has been returned by a successful prior DL-CREATE request primitive that created a queue, which has not yet been deleted.

20.7.2.4 QoS parameters

The DLS-user may specify values for some of the QoS parameters that apply to connectionless data transmission. If a value is not specified, the default value set by DL-management is applied for the DLSAP

20.7.2.5 DL Service subtype

This parameter specifies service subtype of the DLSAP, and the value conveyed in this parameter is as follows:

a) “UUS” -- Unacknowledged Unitdata transfer Service
b) “AUS” -- Acknowledged Unitdata transfer Service
c) “ASS” -- Acknowledged Sequence of unitdata transfer Service
d) “MUS” – Multipoint Unitdata transfer Service
e) “MSS” – Multipoint Sequence of unitdata transfer Service

The function of each service subtype is specified in clause 3.

20.7.2.6 DLL maximum confirm delay

This parameter specifies the value of the maximum confirm delay.

20.7.2.7 DLL priority

This parameter specifies the value of the DLL priority.

20.7.2.8 Authentication level

This parameter specifies the level of the DLL authentication.

20.7.2.9 Maximum residual error rate

This parameter specifies a value of the permissible residual error rate.

20.7.2.10 Transmission Timing Window

This parameter specifies the timing window within the macro cycle. It consists of the starting time in the macro cycle and the time duration when DLE is permitted to transmit DL-UNITDATA DLPDUs.

20.7.2.11 Status

This parameter allows the DLS-user to determine whether the requested DLS was provided successfully, or failed for the reason specified. The possible value conveyed in this parameter is as follows:

a) “success”;
b) “failure — insufficient resources”;
c) “failure — DLSAP-address invalid or unavailable”;
d) “failure — invalid queue binding”;
e) “failure — requested QoS attribute is not available”; or
f) “failure — reason unspecified”.

NOTE Addition to, or refinement of, this list of values, to convey more specific diagnostic and management information is permitted.

20.7.2.12 DLSAP-address DL-identifier

The DLSAP-address DL-identifier parameter is present when the status parameter indicates that the DL-BIND request primitive was successful. The DLSAP-address DL-identifier parameter gives the local DLS-user a means of referring to the DLSAP-address in input parameters of other local DLS primitives that convey the name of the DLSAP-address from the local DLS-user to the local DLE.

The naming-domain of the DLSAP-address DL-identifier is the DL-local-view.

20.8 Unbind

20.8.1 Function

The unbind DLSAP-address DLS primitive is used to unbind a DLSAP-address from the invoking DLSAP. Any queues that were bound explicitly to the DLSAP-address are also unbound from that DLSAP-address at the same time.

20.8.2 Types of parameter

Table 93 lists the primitive and parameter of the unbind DLSAP-address DLS.

Table 93 – DLSAP management UNBIND primitive and parameter

DL-UNBIND	Request	
Parameter name	input	output
DLSAP-address DL-identifier	M	
Status		M

20.8.2.1 DLSAP-address DL-identifier

This parameter specifies the local DL-identifier returned by a successful prior DL-BIND request primitive. The DLS-provider shall unbind the local DL-identifier, its associated DLSAP-address, and any associated queues from the invoking DLSAP, and terminate all outstanding DL-UNITDATA requests associated with that DLSAP-address.

20.8.2.2 Status

This parameter allows the DLS-user to determine whether the requested DLS was provided successfully, or failed for the reason specified. The possible value conveyed in this parameter is as follows:

a) “success”;

b) “failure – reason unspecified”.

NOTE Addition to, or refinement of, this list of values, to convey more specific diagnostic and management information is permitted.

21 Connectionless-mode Data Link Service

21.1 Overview

This clause provides a conceptual definition of the services provided by the DLS-provider to the DLS-user(s). Nothing in this section shall be construed to constrain the actual implementations of the interactions at the DLS-provider to DLS-user interface.

The DLS provides two types of data transfer services;

a) Point-to-point Data Transfer Services

b) Multipoint Data Transfer Services

There are three types of Point-to-point Data Transfer Services;

a) Unacknowledged Unitdata transfer Service (UUS)

b) Acknowledged Unitdata transfer Service (AUS)

c) Acknowledged Sequence of unitdata transfer Service (ASS)

There are two types of Multipoint Data Transfer Service;

a) Multipoint Unitdata transfer Service (MUS)

b) Multipoint Sequence of unitdata transfer Service (MSS)

21.2 Facilities of the Connectionless-mode Data Link Service

The DLS provides the following facilities to DLS-user:

a) a means of transferring DLSDUs with limited length from one source DLSAP to a destination DLSAP or a group of DLSAPs, without establishing or later releasing a Data Link connection. The transfer of DLSDUs is transparent, which means that the boundaries of DLSDUs and the contents of DLSDUs are preserved unchanged by the DLS, and there are no constraints on the DLSDU content (other than limited length) imposed by the DLS provider. QoS for this transmission may be selected by the sending DLS-user.

b) a means by which the status of delivery to that destination DLSAP can be returned to the source DLSAP.

21.3 Model of the Data Link Service

21.3.1 General

This clause uses the abstract model for a layer service defined in ISO/IEC 10731, Clause 5. The model defines local interactions between the DLS-user and the DLS-provider that take place at a DLSAP. Information is passed between the DLS-user and the DLS-provider by DLS primitives that convey parameters.

A defining characteristic of Data Link connectionless-mode unitdata transmission is basically the independent nature of each invocation of the DLS.

21.3.2 DLS-instance identification

A local identification mechanism is provided for each use of the DLS which needs to correlate a confirmation or subsequent cancellation request with its associated request.

DL-connectionless-mode service primitives may be used to transmit independent DLSDUs from one DLSAP to another DLSAP or other DLSAPs. Each DLSDU is transmitted in a single DLPDU. The DLSDU is independent in the sense that it bears no relationship to any other DLSDU transmitted through an invocation of the DLS. The DLSDU is self-contained in that all the information required to deliver the DLSDU is presented to the DLS provider, together with the user data to be transmitted, in a single service access.

21.3.3 Unacknowledged Unitdata transfer Service (UUS)

This service permits a local DLS-user to transfer a DLSDU to a single remote end node. The local DLS-user receives a confirmation acknowledging the completion of the transfer, but not whether the DLSDU was duly received. At the remote end node this DLSDU is delivered to a single local DLS-user, if the respective DLPDU is received error-free. There is no confirmation to the sending DLS-user that such an intended delivery has taken place.

21.3.4 Acknowledged Unitdata transfer Service (AUS)

This service permits the local DLS-user to send a DLSDU to a single remote end node. At the remote end node, the DLSDU is delivered by the remote DLE to its local DLS-user, if the respective DLPDU is transferred error-free. The originating local DLS-user receives a confirmation that indicates whether the DLSDU was received by the remote DLS-user. If an error occurred during the transfer, the originating DLE retransmits the DLSDU appropriately. If the specified number of retransmissions fail, a confirmation with error status will be issued to the local DLS-user.

21.3.5 Acknowledged Sequence of unitdata transfer Service (ASS)

This service permits the local DLS-user to send a sequence of DLSDUs to a single remote end node. At the remote end node, the DLSDU are delivered in the original order requested by the remote DLE to its local DLS-user. The originating local DLS-user receives confirmation of completion of the transmission of each DLSDU, but does not receive confirmation of receipt of the DLSDU by the remote DLS-user. If an error occurred during transfer, the originating DLE retransmits the DLSDUs appropriately. A sequence of DLSDUs is formed from the requested DLSDUs by originating DLE according to a preconfigured rule.

21.3.6 Multipoint Unitdata transfer Service (MUS)

This service permits a local DLS-user to transfer a DLSDU to the preconfigured group of remote end nodes at the same time. The local DLS-user receives confirmation of completion of the transmission, but does not receive confirmation of receipt of the DLSDU by the remote DLS-user. At each addressed remote end node, this DLSDU is delivered to the remote DLS-user, if the respective DLPDU is received error-free. There is no confirmation to the sending DLS-user that such an intended delivery has taken place.

21.3.7 Multipoint Sequence of unitdata transfer Service (MSS)

This service permits a local DLS-user to transfer a DLSDU to a preconfigured group of remote end nodes at the same time. The local DLS-user receives confirmation of completion of the transmission, but does not receive confirmation of receipt of the DLSDU by the remote DLS-user. At each remote addressed end node, the DLSDU are delivered to the remote DLS-user in the original requested order.

21.4 Quality of service

A DLS-user may select many aspects of the various Data Link Services. The term “Quality of Service” (QoS) refers to certain characteristics of a connectionless-mode data transmission as observed between the DLSAPs. QoS describes aspects of data transmission that are related solely to the DLS-provider. QoS can only be determined properly when DLS-user behaviour does not constrain or impede performance of the DLS.

The QoS attributes of the DL-data transfer services apply conceptually to the connectionless operation. The DLS-user may specify values for these attributes when binding a DLSAP-address to the DLS-user's DLSAP; any unspecified attributes will assume the default values set by DL-management.

21.4.1 Data delivery features

The data delivery feature depends on the type of service, as specified in Table 94.

Table 94 – Data delivery features of each type of service

Service Type	Action on lost DLSDUs	Action on duplicated DLSDUs	Action on misordered DLSDUs	Action on lack of receive buffer
UUS	not detected	delivered as received	delivered as received	discarded
AUS	recovered	discarded	delivered as received	wait & retransmitted
ASS	recovered conditionally	discarded	delivered in order	wait & retransmitted
MUS	not detected	delivered as received	delivered as received	discarded
MSS	detected	discarded	delivered as received	discarded