IEC 61158-6-21

Edition 1.0 2010-08

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-21: Application layer protocol specification – Type 21 elements

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

About the IEC

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

About IEC publications

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

- Catalogue of IEC publications: www.iec.ch/searchpub

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 It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub

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

- Electropedia: www.electropedia.org

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

- Customer Service Centre: www.iec.ch/webstore/custserv

If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

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

IEC 61158-6-21

Edition 1.0 2010-08

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-21: Application layer protocol specification – Type 21 elements

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XA

ICS 25.04.40; 35.100.70; 35.110

ISBN 978-2-88912-135-9

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-21: Application layer protocol specification – Type 21 elements

FOREWORD

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

NOTE 1 Use of some of the associated protocol types is restricted by their intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a particular data-link layer protocol type to be used with physical layer and application layer protocols in type combinations as specified explicitly in the IEC 61784 series. Use of the various protocol types in other combinations may require permission of their respective intellectual-property-right holders.

International Standard IEC 61158-6-21 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This standard cancels and replaces IEC/PAS 62573 published in 2008. This first edition constitutes a technical revision

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

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

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

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

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

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

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

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

INTRODUCTION

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

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-21: Application layer protocol specification – Type 21 elements

1 Scope

1.1 General

This standard is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the three-layer fieldbus reference model described in IEC/TR 61158-1:2010.

This standard contains material specific to the Type 21 communication protocol.

1.2 Overview

The Fieldbus Application Layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a window between corresponding application programs.

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment, as well as material specific to Type 21. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions must to be completed with some defined level of certainty. Failure to complete specified actions within the required time risks the failure of the applications requesting the actions, with attendant risk to equipment, plant, and possibly human life.

This standard defines interactions between remote applications. It also defines the externally visible behavior provided by the Type 21 application layer in terms of:

a) the formal abstract syntax defining the application layer protocol data units (APDUs) conveyed between communicating application entities;

b) the transfer syntax defining encoding rules that are applied to the APDUs;

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

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

The purpose of this standard is to:

a) describe the wire-representation of the service primitives defined in IEC 61158-5-21:2010;

b) describe the externally visible behavior associated with their transfer.

This standard defines the protocol of the Type 21 application layer in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI application layer structure (ISO/IEC 9545).

1.3 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the Type 21 application layer services.

A secondary objective is to provide migration paths from previously existing industrial communications protocols.

1.4 Conformance

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

2 Normative references

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

IEC 61158-3-21:2010 [1] , *Industrial communication networks – Fieldbus specifications – Part 3-21: Data-link layer service definition – Type 21 elements*

IEC 61158-4-21:2010[1], *Industrial communication networks – Fieldbus specifications – Part 4-21: Data-link layer protocol specification – Type 21 elements*

IEC 61158-5-21:2010[1], *Industrial communication networks – Fieldbus specifications – Part 5-21: Application layer service definition – Type 21 elements*

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

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

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

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

ISO/IEC 10731:1994, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ISO/IEC 9899, *Programming Languages – C*

IEEE 754-2008, *IEEE Standard for Binary Floating-Point Arithmetic*

[1] To be published.

3 Terms, definitions, symbols, abbreviations, and conventions

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 ISO/IEC 7498-1 terms

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

a) application entity
b) application process
c) application protocol data unit
d) application service element
e) application entity invocation
f) application process invocation
g) application transaction
h) real open system
i) transfer syntax

3.1.2 ISO/IEC 8822 terms

For the purposes of this document, the following terms as defined in ISO/IEC 8822 apply:

a) abstract syntax
b) presentation context

3.1.3 ISO/IEC 8824-1 terms

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

a) object identifier
b) type

3.1.4 ISO/IEC 9545 terms

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

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

3.2 Other terms and definitions

3.2.1
application
function or data structure for which data are consumed or produced

3.2.2
application objects
multiple object classes that manage and provide a runtime exchange of messages across the network and within the network device

3.2.3
application process
part of a distributed application on a network, which is located on one device and addressed unambiguously

3.2.4
application process identifier
distinguishes multiple application processes used in a device

3.2.5
application process object
component of an application process that is identifiable and accessible through an FAL application relationship

NOTE Application process object definitions are composed of a set of values for the attributes of their class (see the definition for “application process object class”). Application process object definitions may be accessed remotely using the services of the FAL Object Management ASE. FAL Object Management services can be used to load or update object definitions, to read object definitions, and to create and delete application objects and their corresponding definitions dynamically.

3.2.6
application process object class
class of application process objects defined in terms of the set of their network-accessible attributes and services

3.2.7
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation

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

3.2.8
application relationship application service element
application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.2.9
application relationship endpoint
context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship

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

3.2.10
attribute
description of an externally visible characteristic or feature of an object

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

3.2.11
behavior
indication of how an object responds to particular events

3.2.12
channel
single physical or logical link of an input or output application object of a server to the process

3.2.13
class
set of objects, all of which represent the same type of system component

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

3.2.14
class attributes
attribute shared by all objects within the same class

3.2.15
class code
unique identifier assigned to each object class

3.2.16
class-specific service
service defined by a particular object class to perform a required function that is not performed by a common service

NOTE A class-specific object is unique to the object class that defines it.

3.2.17
client
a) object that uses the services of another (server) object to perform a task

b) initiator of a message to which a server reacts

3.2.18
consume
act of receiving data from a producer

3.2.19
consumer
node or sink that receives data from a producer

3.2.20
consuming application
application that consumes data

3.2.21
conveyance path
unidirectional flow of APDUs across an application relationship

3.2.22
cyclic
repetitive in a regular manner

3.2.23
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.2.24
device
physical hardware connected to the link

NOTE A device may contain more than one node.

3.2.25
device profile
a collection of device-dependent information and functionality providing consistency between similar devices of the same device type

3.2.26
diagnostic information
all data available at the server for maintenance purposes

3.2.27
end node
producing or consuming node

3.2.28
endpoint
one of the communicating entities involved in a connection

3.2.29
error
discrepancy between a computed, observed, or measured value or condition and the specified or theoretically correct value or condition

3.2.30
error class
general grouping for related error definitions and corresponding error codes

3.2.31
error code
identification of a specific type of error within an error class

3.2.32
event
instance of a change of conditions

3.2.33
FIFO variable
variable object class composed of a set of homogeneously typed elements, where the first written element is the first element that can be read

NOTE In a fieldbus system, only one complete element can be transferred as a result of one service invocation.

3.2.34
frame
simplified synonym for data link protocol data unit (DLPDU)

3.2.35
group
a) (General): a general term for a collection of objects
b) (Addressing): when describing an address, an address that identifies more than one entity

3.2.36
invocation
act of using a service or other resource of an application process

NOTE Each invocation represents a separate thread of control that may be described by its context. Once the service completes, or use of the resource is released, the invocation ceases to exist. For service invocations, a

service that has been initiated but not yet completed is referred to as an outstanding service invocation. For service invocations, an Invoke ID may be used to identify the service invocation unambiguously and differentiate it from other outstanding service invocations.

3.2.37
index
address of an object within an application process

3.2.38
instance
actual physical occurrence of an object within a class that identifies one of many objects in the same object class

EXAMPLE California is an instance of the object class US-state.

NOTE The terms object, instance, and object instance are used to refer to a specific instance.

3.2.39
instance attributes
attribute that is unique to an object instance and not shared by the object class

3.2.40
instantiated
object that has been created in a device

3.2.41
logical device
specific FAL class that abstracts a software component or a firmware component as an autonomous self-contained facility of an automation device

3.2.42
manufacturer ID
identification of each product manufacturer by a unique number

3.2.43
management information
network-accessible information that supports management of the operation of the fieldbus system, including the application layer

NOTE Managing includes functions, such as controlling, monitoring, and diagnosis.

3.2.44
network
set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers, and lower-layer gateways

3.2.45
object
abstract representation of a particular component within a device, usually a collection of related data in the form of variables, and methods (procedures) for operating on that data that have clearly defined interface and behavior

3.2.46
object dictionary
collection of definitions, communication-specific attributes and parameters, and application-dependent data

3.2.47
object-specific service
service unique to the object class that defines it

3.2.48
physical device
automation or other network device

3.2.49
point-to-point connection
connection that exists between exactly two application objects

3.2.50
pre-established AR endpoint
AR endpoint placed in an established state during configuration of the AEs that control its endpoints

3.2.51
process data
object(s) that are already pre-processed and transferred cyclically for the purpose of information or further processing

3.2.52
produce
act of sending data to be received by a consumer

3.2.53
producer
node that is responsible for sending data

3.2.54
property
general term for descriptive information about an object

3.2.55
provider
source of a data connection

3.2.56
publisher
role of an AR endpoint that transmits APDUs onto the fieldbus for consumption by one or more subscribers

NOTE A publisher may not be aware of the identity or number of subscribers.

3.2.57
publishing manager
role of an AR endpoint in which it issues one or more confirmed service request application protocol data units (APDUs) to a publisher to request that a specified object be published. Two types of publishing managers are defined by this standard, pull publishing managers and push publishing managers, each of which is defined separately.

3.2.58
push publisher
type of publisher that publishes an object in an unconfirmed service request APDU

3.2.59
push publishing manager
type of publishing manager that requests that a specified object be published using an unconfirmed service

3.2.60
push subscriber
type of subscriber that recognizes received unconfirmed service request APDUs as published object data

3.2.61
sending
service user that sends a confirmed primitive or an unconfirmed primitive, or a service provider that sends a confirmed APDU or an unconfirmed APDU

3.2.62
server
a) role of an application relationship endpoint (AREP) in which it returns a confirmed service response APDU to the client that initiated the request

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

3.2.63
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.2.64
station
host of one AP, identified by a unique data link connection endpoint (DLCEP)-address

3.2.65
subscriber
role of an AREP in which it receives APDUs produced by a publisher

3.2.66
receiving
service user that receives a confirmed primitive or an unconfirmed primitive, or a service provider that receives a confirmed APDU or an unconfirmed APDU

3.2.67
resource
processing or information capability of a subsystem

3.3 Abbreviations and symbols

AE	Application Entity
AL	Application Layer
ALME	Application Layer Management Entity
ALP	Application Layer Protocol
APO	Application Object
AP	Application Process
APDU	Application Protocol Data Unit
AR	Application Relationship
AREP	Application Relationship End Point
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Cnf	Confirmation
DL-	(as a prefix) Data Link -
DLCEP	Data Link Connection End Point

DLL	Data Link Layer
DLM	Data Link Management
DLSAP	Data Link Service Access Point
DLSDU	DL-service-data-unit
DNS	Domain Name Service
FAL	Fieldbus Application Layer
Ind	Indication
Req	Request
Rsp	Response

3.4 Conventions

3.4.1 General conventions

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

This standard uses the descriptive conventions given in IEC 61158-6-21:2010 for FAL service definitions.

3.4.2 Convention for the encoding of reserved bits and octets

The term “reserved” may be used to describe bits in octets or whole octets. All bits or octets that are reserved should be set to zero on the sending side. They will not be tested on the receiving side except if explicitly stated, or if the reserved bits or octets are checked by a state machine.

The term “reserved” may also be used to indicate that certain values within the range of a parameter are reserved for future extensions. In this case the reserved values should not be used at the sending side. They shall not be tested at the receiving side except if explicitly stated, or if the reserved values are checked by a state machine.

3.4.3 Conventions for the common coding of specific field octets

APDUs may contain specific fields that carry information in a primitive and condensed way. These fields shall be coded in the order according to Figure 1.

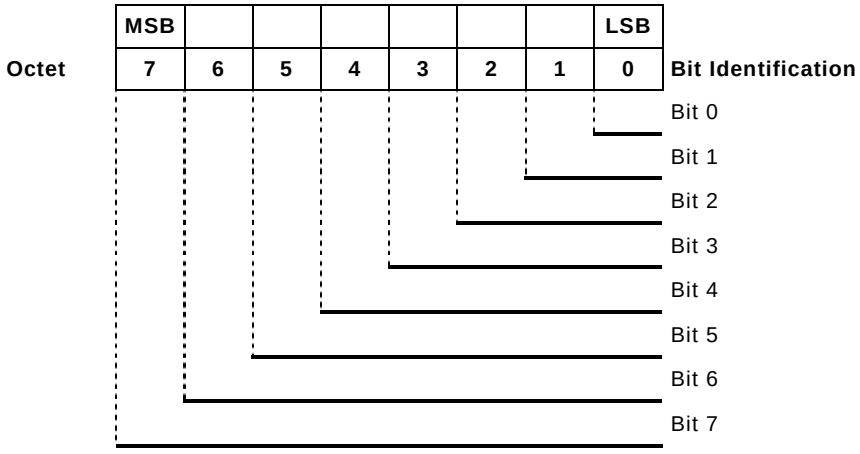

Figure 1 – Common structure of specific fields

Bits may be grouped. Each bit or group of bits shall be addressed by its bit identification (*e.g.*, Bit 0, Bits 1–4). The position within the octet shall be as shown in Figure 1. Alias names may be used for each bit or group of bits, or they may be marked as reserved. The grouping of

individual bits shall be in ascending order without gaps. The values for a group of bits may be represented as binary, decimal, or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 8 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest group number represents the most significant bit (MSB) of the grouped bits.

EXAMPLE: Description and relation for the specific field octet

Bit 0: reserved.

Bits 1–3: Reason_Code. The decimal value 2 for the Reason_Code means general error.

Bits 4–7: Always set to one.

The octet that is constructed according to the description above looks as follows:

(MSB) Bit 7 = 1;

Bit 6 = 1;

Bit 5 = 1;

Bit 4 = 1;

Bit 3 = 0;

Bit 2 = 1;

Bit 1 = 0;

(LSB) Bit 0 = 0.

This bit combination has an octet value representation of 0xf4.

3.4.4 Conventions for APDU abstract syntax definitions

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

3.4.5 Conventions for APDU transfer syntax definitions

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

3.4.6 Conventions for AE state machine definitions

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

Table 1 – Conventions used for AE state machine definitions

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

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

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

xxx Parameter name.

Example:

Identifier := Reason

means value of the Reason parameter is assigned to the parameter called Identifier.

“xxx” Indicates a fixed value.

Example:

Identifier := “abc”

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

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

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

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

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

&& Logical “AND”

|| Logical “OR”

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

for

endfor

if

else

elseif

The following shows examples of description using the reserved words.

Example 1:

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

Example 2:

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

4 FAL syntax description

4.1 General

This description of the Type 21 abstract syntax uses formalisms similar to ASN.1, although the encoding rules differ from that standard.

4.2 FAL-AR PDU abstract syntax

4.2.1 Top level definition

```
APDU ::= CHOICE {
    ConfirmedSend-CommandPDU
    ConfirmedSend-ResponsePDU
    UnconfirmedSend-CommandPDU
}
```

4.2.2 Confirmed send service

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

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

4.2.3 Unconfirmed send service

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

4.2.4 FalArHeader

```
FalArHeader ::= Unsigned8 {
  -- bit 8-7        ProtocolVersion
  -- bit 6-4        Protocol Identifier        Identifiers abstract syntax revision, and encoding rules
   --bit 3-1        PDU Identifier             Identifies a PDU type within a Protocol Identifier
}
```

4.2.5 InvokeID

```
InvokeID ::= Unsigned8                        Identifies this invocation of the service
```

4.2.6 ServiceType

```
ServiceType ::= Unsigned16                    Contains the context specific tags for the PDU body
```

4.3 Abstract syntax of PDU body

4.3.1 ConfirmedServiceRequest PDUs

```
ConfirmedServiceRequest::= CHOICE {
      Read-Request                  [0]    IMPLICIT      Read-RequestPDU,
      Write-Request                 [1]    IMPLICIT      Write-RequestPDU,
      Identify-Request              [2]    IMPLICIT      Identify-RequestPDU,
      Status-Request                [3]    IMPLICIT      Status-RequestPDU,
}
```

4.3.2 ConfirmedServiceResponse PDUs

```
ConfirmedServiceRequest::= CHOICE {
      Read-Response                 [0]    IMPLICIT      Read-ResponsePDU,
      Write-Response                [1]    IMPLICIT      Write-ResponsePDU,
      Identify-Response             [2]    IMPLICIT      Identify-ResponsePDU,
      Status-Response               [3]    IMPLICIT      Status-ResponsePDU,
}
```

4.3.3 UnconfirmedServiceRequest PDUs

```
UnconfirmedServiceRequest::= CHOICE  {
      TB-Request              [0]      IMPLICIT  TB-transferPDU
      COS-Request             [1]      IMPLICIT  COS-transferPDU

}
```

4.3.4 Error information

4.3.4.1 Error type

```
ConfirmedServiceRequest::= CHOICE  {
      Service status                [0]    IMPLICIT      ErrorClass,
      Status code                   [1]    IMPLICIT      Integer16 OPTIONAL,
}
```

4.3.4.2 Error class

```
   ErrorClass ::= CHOICE {
      noError                              [0]    IMPLICIT      Integer8 {
```

```
                                    normal                          (0),
                                    Other                           (1)
        }
        Definition              [1] IMPLICIT    Integer8 {
                                    other                           (0),
                                    object-undefined                (1),
        }
        Resource                [2] IMPLICIT    Integer8 {
                                    other                           (0),
                                    memory-unavailable              (1)
        }
        Service                 [3] IMPLICIT    Integer8 {
                                    other                           (0),
                                    pdu-size                        (1),
                                    illegal-parameter               (2)
        }
        Access                  [4] IMPLICIT    Integer8 {
                                    other                           (0),
                                    object-access-denied            (1),
                                    invalid-address                 (2),
                                    object-access-unsupported       (3),
                                    object-non-existent             (4),
        }
        Other                   [5] IMPLICIT    Integer8 {
                                    other                           (0)
        }
    }
```

4.3.4.3 Status code

The status codes for the confirmed response primitive are listed in Table 2.

Table 2 – Status code for the confirmed response primitive

Error Code	Error Type	Error details and causes
0x01	Service type Error	Unknown Service type
0x02	Object Identifier Error	Object-access-unsupported
0x03	Data type error	Other data type than supported.
0x04	Object Offset Error	Request exceeds the area each object supports.
0x05	Data Length error	Request exceeds the maximum range of Ethernet rules to read or write at a time.

4.4 Protocol data units (PDUs) for application service elements (ASEs)

4.4.1 PDUs for Application process ASE

4.4.1.1 Identify-Request PDUs

```
Identify-Request PDU::= NULL
```

4.4.1.2 Identify-Response PDUs

```
Identify-ResponsePDU ::= SEQUENCE {
    DL–entity identifier
    MAC address
    Port information
    Protocol version
    Device type
    Device description
    Hardware-Version
    Serial Number
    Software-Version
    Software-Date
    Vendor ID
    Product Code
}
```

4.4.1.3 Status-Request PDUs

Status-Request PDU::= NULL

4.4.1.4 Status-Response PDUs

```
Status-ResponsePDU ::= SEQUENCE {
    Device flags
    Device state
    tx_cnt_normal
    tx_cnt_all
    rx_cnt_normal
    rx_cnt_all
    relay_cnt_normal
    relay_cnt_all
}
```

4.4.1.5 DL–entity identifier

dl-address ::= Unsigned16 — See IEC 61158-4-21:2010, 4.6.5.2.

4.4.1.6 MAC address

MAC address::= Unsigned48 — See IEC 61158-4-21:2010, 4.6.5.8.

4.4.1.7 Port information

Port information::= Unsigned16 — See IEC 61158-4-21:2010, 4.6.5.9.

4.4.1.8 Protocol version

Protocol version ::= Unsigned8 — See IEC 61158-4-21:2010, 4.6.5.10.

4.4.1.9 Device type

Device type::= Unsigned16 — See IEC 61158-4-21:2010, 4.6.5.11.

4.4.1.10 Device description

Device description ::= VISIBLE_STRING[16] — See IEC 61158-4-21:2010, 4.6.5.12.

4.4.1.11 Hardware-Version

Hardware-Version ::= Unsigned16 — Contains the hardware version of the device.

4.4.1.12 Serial Number

Serial Number ::= Unsigned16 — Contains the serial number of the device.

4.4.1.13 Software-Version

Software-Version ::= Unsigned16 — Contains the software version of the device.

4.4.1.14 Vendor ID

Vendor ID ::= Unsigned16 — Contains the vendor ID of the device.

4.4.1.15 Product Code

Product Code ::= Unsigned16 — Contains the product code of the device.

4.4.1.16 Device flags

Device flags ::= Unsigned16 — See IEC 61158-4-21:2010, 4.6.5.3.

4.4.1.17 Device state

Device state ::= Unsigned16 — See IEC 61158-4-21:2010, 4.6.5.4.

4.4.1.18 tx_cnt_normal

tx_cnt_normal ::= Unsigned32 — The value of the transmit count of normal packets.

4.4.1.19 tx_cnt_all

tx_cnt_all ::= Unsigned32 — The value of the transmit count of both error packets and

normal packets.

4.4.1.20 rx_cnt_normal

rx_cnt_normal ::= Unsigned32 — The value of the receive count of normal packets.

4.4.1.21 rx_cnt_all

rx_cnt_all ::= Unsigned32 — The value of the receive count of both error packets and normal packets.

4.4.1.22 relay_cnt_normal

relay_cnt_normal ::= Unsigned32 — The value of the relay count of normal packets.

4.4.1.23 relay_cnt_all

relay_cnt_all ::= Unsigned32 — The value of the relay count of both error packets and normal packets.

4.4.2 PDUs for Service data object ASE

4.4.2.1 Read service PDUs

```
Read-RequestPDU::= SEQUENCE {
   Object List Count
   Object identifier (k)
   Data type
   offset
   length
   Object identifier (m)
   Data type
   offset
   length
   ...                                  — (further read requests)
}
Read-ResponsePDU ::= SEQUENCE {
   Service status
   Object List Count
   Object identifier (k)
   Data type
   offset
   length
   data
   Object identifier (m)
   Data type
   offset
   length
   data
   ...                                  — (further read responses)
}
```

4.4.2.2 Write service PDUs

```
Write-RequestPDU::= SEQUENCE {
   Object List Count
   Object identifier (k)
   Data type
   offset
   length
   data
   Object identifier (m)
   Data type
   offset
   length
   data
   ...                                  — (further write requests)
}

Write-ResponsePDU ::= SEQUENCE {
   Service status
   Status code                          — (optional)
}
```

4.4.2.3 Object List Count

Object list count ::= Unsigned16 — Number of objects.

4.4.2.4 Object identifier

Object identifier ::= Unsigned16 — Specifies an entry of the device object.

4.4.2.5 Data type

Data type ::= Unsigned16 — Contains the numeric identifier of the data type.

4.4.2.6 Offset

offset ::= Unsigned32 — Provides the offset related to the start of the object.

4.4.2.7 Length

length ::= Unsigned32 — The number of octets of the service which are to be read or written.

4.4.2.8 data

data ::= Any — Application-dependent type and length;
total frame length shall comply with Ethernet rules.

4.4.3 PDUs for Process data object ASE

4.4.3.1 TB-transfer service PDUs

```
TB-transfer PDU ::= SEQUENCE {
      TBArep,      -- Block number
      TBData       -- content of TB segment
}
```

4.4.3.2 TBArep

TBArep ::= Unsigned16 — Block number

4.4.3.3 TBData

```
TBData ::= SEQUENCE {
   blen     Unsigned16,          — TB octet length
   payload-data::=Any            — TB content
```

4.4.3.4 COS-transfer service PDUs

```
COS-transfer PDU ::= SEQUENCE {
      COSArep,    -- Block number
      COSData     -- content of COS segment
}
```

4.4.3.5 COSArep

COSArep ::= Unsigned16 — Block number

4.4.3.6 COSData

```
COSData ::= SEQUENCE {
   blen     Unsigned16,          — COS octet length
   payload-data::=Any            — COS content
```

5 Transfer Syntax

5.1 Overview of encoding

The encoded FAL-PDUs encoded shall have a uniform format. They shall consist of two major parts: the APDU header part and the APDU body part as shown in Figure 2.

(1)	(1)	(2)	(n) --- octets
FalArHeader Field	(InvokeID)	ServiceType	Service Specific Parameters
<---------- APDU header ---------->			<---------- APDU body ---------->

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

Figure 2 – 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 DLL service provides a DLSDU parameter that implies the length of the APDU. Thus, the APDU length information is not included in the APDU.

5.2 APDU header encoding

The APDU Header part is always present in all APDUs that conform to this standard. It consists of three fields: the FalArHeader Field, the InvokeID Field, and the ServiceType Field, as shown in Figure 2.

5.2.1 Encoding of FalArHeader field

All the FAL PDUs have the common PDU-header called FalArHeader. The FalArHeader identifies abstract syntax, transfer syntax, and each of the PDUs. Table 3 defines how this header shall be used.

Table 3 – Encoding of FalArHeader field

Bit position of the FalArHeader			PDU type	Protocol version
8 7	6 5 4	3 2 1		
01	001	000	ConfirmedSend-CommandPDU	Version 1
01	001	100	ConfirmedSend-ResponsePDU	Version 1
01	010	000	UnconfirmedSend-CommandPDU	Version 1
NOTE All other code points are reserved for additional protocols and future revisions.				

5.2.2 Encoding of InvokeID Field

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

5.2.3 Encoding of Type field

The service type of an APDU is encoded in the Type field that is always the third octet of the APDU.

All bits of the Type field are used to encode the service type, as follows:

a) the service types shall be encoded in bits 16 to 1 of the Type field, with bit 16 the MSB and bit 1 the LSB. The range of service type shall be in the range 0 to 65 534;

b) the value of 65 535 is reserved for future extensions to this standard;

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

Figure 3 illustrates the encoding of the Type field.

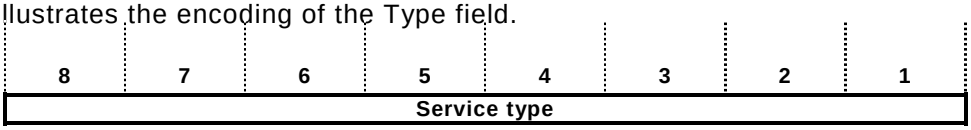

Figure 3 – Type field

5.3 APDU body encoding

5.3.1 General

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

a) identifier octet;
b) length octet(s);
c) contents octet(s).

NOTE Identification octet and content length octets do not exist in Type 21.

5.4 Encoding of Data types

5.4.1 General description of data types and encoding rules

The format of this data and its meaning shall be known by the producer and consumer(s) to be able to exchange meaningful data. This standard model uses the concept of data types to achieve this.

The encoding rules define the representation of values of data types and the transfer syntax for the representations. Values are represented as bit sequences. Bit sequences are transferred in sequences of octets (. For numerical data types, the encoding is little-endian style as shown in Table 2.

5.4.2 Transfer syntax for bit sequences

A bit sequence is reordered into a sequence of octets for transmission. Hexadecimal notation is used for octets as specified in ISO/IEC 9899. Let $b = b_0 ... b_{n-1}$ be a bit sequence and k be a non-negative integer such that $8(k-1) < n \leq 8k$. Then, b is transferred in k octets assembled as shown in Table 4. The bits b_i, $i \geq n$ of the highest numbered octet are do-not-care bits.

Table 4 – Transfer Syntax for bit sequences

octet number	1.	2.	k.
	b_0 .. b_7	b_8 .. b_{15}	b_{8k-8} .. b_{8k-1}

Octet 1 is transmitted first and octet *k* is transmitted last. The bit sequence is transferred as follows across the network (transmission order within an octet is determined by ISO/IEC 8802-3:2000):

$b_0, b_1, ..., b_7, b_8, ..., b_{15}, ...$

EXAMPLE

Bit 0	...	Bit 9
0011b	1000b	01b
Ch	1h	2h
		= 21Ch

The bit sequence $b = b_0 .. b_9 = 0011\ 1000\ 01_b$ represents an Unsigned10 with the value 21Ch, and is transferred in two octets: first 1Ch and then 02h.

5.4.3 Encoding of a Boolean value

Data of basic data type BOOLEAN can have the values TRUE or FALSE.

The values are represented as bit sequences of length 1. The value TRUE is represented by 1, and FALSE by 0.

A BOOLEAN shall be transferred over the network as UNSIGNED8 of value 1 (TRUE) or 0 (FALSE). Sequences of BOOLEANs may be packed into one UNSIGNED8. Sequences of BOOLEAN and BIT type items may be also packed into one UNSIGNED8.

5.4.4 Encoding of an unsigned integer value

Data of basic data type UNSIGNEDn has values in the non-negative integers. The value range is 0, ..., 2^n-1. The data are represented as bit sequences of length n.

The bit sequence

$b = b_0 \ldots b_{n-1}$

is assigned value defined by

$UNSIGNEDn(b) = b_0 \times 2^0 + b_1 \times 2^1 + \ldots + b_{n-1} \times 2^{n-1}$

Note that the bit sequence starts on the left with the least significant byte.

Example: The value $266_d = 10A_h$ with data type UNSIGNED16 is transferred in two octets across the bus, first $0A_h$ and then 01_h.

The UNSIGNEDn data types are transferred as shown in Table 5.

Table 5 – Transfer syntax for data type UNSIGNEDn

octet number	0	1	2	3	4	5	6	7
UNSIGNED8	$b_0..b_7$							
UNSIGNED16	$b_0..b_7$	$b_8..b_{15}$						
UNSIGNED32	$b_0..b_7$	$b_8..b_{15}$	$b_{16}..b_{23}$	$b_{24}..b_{31}$				
UNSIGNED64	$b_0..b_7$	$b_8..b_{15}$	$b_{16}..b_{23}$	$b_{24}..b_{31}$	$b_{32}..b_{39}$	$b_{40}..b_{47}$	$b_{48}..b_{55}$	$b_{56}..b_{63}$

5.4.5 Encoding of a signed integer

Data of basic data type INTEGERn has values in the integers. The value range is from -2^{n-1} to 2^{n-1}-1. The data are represented as bit sequences of length n. The bit sequence

$b = b_0 .. b_{n-1}$

is assigned value defined by

$INTEGERn(b) = b_0 \times 2^0 + b_1 \times 2^1 + \ldots + b_{n-2} \times 2^{n-2}$ if $b_{n-1} = 0$

and, performing two's complement arithmetic,

$INTEGERn(b) = -\ INTEGERn(\hat{}b) - 1$ if $b_{n-1} = 1$

Note that the bit sequence starts on the left with the least significant bit.

EXAMPLE: The value $-266_d = 0xFEF6_h$ with data type Integer16 is transferred in two octets, first 0xF6 and then 0xFE.

The INTEGERn data types are transferred as specified Table 6.

Table 6 – Transfer syntax for data type INTEGERn

octet number	0	1	2	3	4	5	6	7
INTEGER8	$b_0..b_7$							
INTEGER16	$b_0..b_7$	$b_8..b_{15}$						
INTEGER32	$b_0..b_7$	$b_8..b_{15}$	$b_{16}..b_{23}$	$b_{24}..b_{31}$				
INTEGER64	$b_0..b_7$	$b_8..b_{15}$	$b_{16}..b_{23}$	$b_{24}..b_{31}$	$b_{32}..b_{39}$	$b_{40}..b_{47}$	$b_{48}..b_{55}$	$b_{56}..b_{63}$

5.4.6 Encoding of a floating point value

Data of basic data types REAL32 and REAL64 have values in real numbers.

The data type REAL32 is represented as a bit sequence of length 32. The encoding of values follows the IEEE 754-2008 standard for single-precision floating point.

The data type REAL64 is represented as a bit sequence of length 64. The encoding of values follows the IEEE 754-2008 standard for double-precision floating point numbers.

A bit sequence of length 32 either has a value (finite non-zero real number, ± 0, ± _) or is not a number (NaN).

The bit sequence

$$b = b_0 \ldots b_{31}$$

is assigned value (finite non-zero number) defined by

$$REAL32(b) = (-1)^S \times 2^{E-127} \times (1+ F)$$

where

$S = b_{31}$ is the sign.

$E = b_{23} \times 2^0 + \ldots + b_{30} \times 2^7$, $0 < E < 255$, is the un-biased exponent.

$F = 2^{-23} \times (b_0 \times 2^0 + b_1 \times 2^1 + \ldots + b_{22} \times 2^{22})$ is the fractional part of the number.

E = 0 is used to represent ± 0. E = 255 is used to represent infinities and NaNs.

The bit sequence shall start on the left with the least significant bit.

5.4.7 Encoding of an octet string value

The data type OCTET_STRINGlength is defined as follows where “length” is the length of the octet string.

ARRAY [length] OF UNSIGNED8 OCTET_STRINGlength

5.4.8 Encoding of a visible string value

VISIBLE_CHAR are 0_h and the range from 20_h to $7E_h$. The data are interpreted as ISO 646-1973(E) 7- bit coded characters, and “length” is the length of the visible string.

UNSIGNED8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VISIBLE_STRINGlength

There is no 0_h necessary to terminate the string.

5.4.9 Encoding of a Unicode string value

The data type UNICODE_STRINGlength is defined below where “length” is the length of the Unicode string.

ARRAY [length] OF UNSIGNED16 UNICODE_STRINGlength

5.4.10 Encoding of a time of day value

The data type TimeOfDay represents absolute time. TimeOfDay is represented as a bit sequence of length 48.

Component “ms” is the time in milliseconds after midnight. Component “days” is the number of days since 1984-01-01.

```
STRUCT OF
   UNSIGNED28     ms,
   VOID4          reserved,
   UNSIGNED16     days
TIME_OF_DAY
```

The encoding is as shown in Figure 4.

bits	7	6	5	4	3	2	1	0	
octets									
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	number of ms since midnight
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	number of days since 1984-01-01
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	msb								

Figure 4 – Encoding of Time of Day value

5.4.11 Encoding of a Time Difference value

The data type TimeDifference represents a time difference or a period of time. TimeDifference is represented as a bit sequence of length 48.

Time differences are sums of numbers of days and milliseconds. Component “ms” is the number of milliseconds. Component “days” is the number of days.

```
STRUCT OF
   UNSIGNED28      ms,
   VOID4           reserved,
   UNSIGNED16      days
TIME_DIFFERENCE
```

The encoding is as shown in Figure 5.

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

Figure 5 – Encoding of Time Difference value

6 FAL protocol state machines

Interfaces to FAL services and protocol machines are specified in this standard.

NOTE The state machines specified in this standard and Application Relationship Protocol Machines defined below only define the valid events for each. Handling invalid events is a local responsibility.

The behavior of the FAL is described by the protocol machines shown in Figure 6. The three types of protocol machine are: FAL service protocol machines (FSPMs), application relationship protocol machines (ARPMs), and DLL mapping protocol machines (DMPMs). Specific sets of these protocol machines are defined for different types of application relationship endpoint (AREP). Figure 6 also shows the primitives exchanged between the protocol machines.

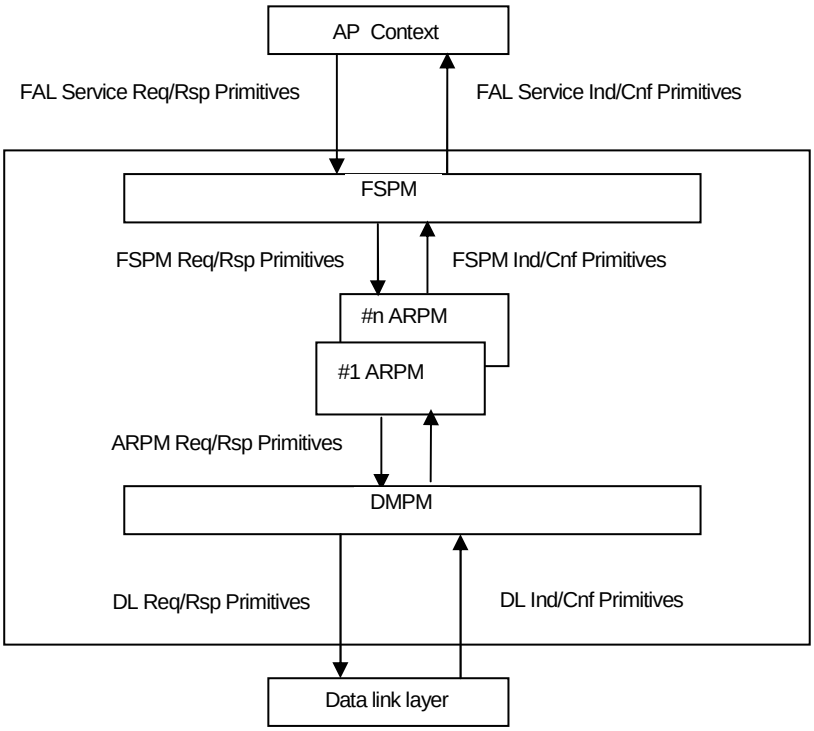

Figure 6 – Primitives exchanged between protocol machines

The FSPM is responsible for the following:

a) accepting service primitives from the FAL service user and converting them into FAL internal primitives;

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

c) accepting FAL internal primitives from the ARPM and converting them into service primitives for the AP context;

d) delivering 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:

a) accepting FAL internal primitives from the FSPM, and creating and sending other FAL internal primitives to either the FSPM or the DMPM, based on the AREP and primitive types;

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

c) to establish or release the specified AR if the primitives are for the Establish or Abort services, respectively.

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) accepting FAL internal primitives from the ARPM, preparing DLL service primitives, and sending them to the DLL;

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

c)

7 AP context state machine

There is no AP context state machine defined for this protocol.

8 FAL service protocol machine

8.1 General

These FSPMs are defined as follows:

a) Application process ASE Protocol Machine (APAM);
b) Service data object ASE Protocol Machine (SDOM);
c) Process data object ASE Protocol Machine (PDOM).

8.2 Common parameters of the primitives

Many services share a group of common parameters. Instead of defining them repeatedly with each individual service, the following common definitions are provided once below.

AREP
This parameter contains sufficient information 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.

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.

Service status
This parameter provides information on the result of service execution. It is returned in all confirmed service response primitives (+ and -).

8.3 AP ASE protocol machine

8.3.1 Primitive definitions

8.3.1.1 Primitives exchanged

Table 7 shows the service primitives, including their associated parameters exchanged between the FAL-user and the APAM.

Table 7 – Primitives exchanged between FAL-user and APAM

Primitive	Source	Associated parameters	Functions
identify.req	FAL-user	AREP InvokeID Service type	this primitive is used to request device information
status.req	FAL-user	AREP InvokeID Service type	this primitive is used to request device status
identify.rsp	FAL-user	AREP InvokeID Service type Service status Identify Data	this primitive is used to respond to an identify request
status.rsp	FAL-user	AREP InvokeID Service type Service status Status Data	this primitive is used to respond to a status request
identify.ind	APAM	AREP InvokeID Service type	this primitive is used to indicated an identify request
status.ind	APAM	AREP InvokeID Service type	this primitive is used to indicate a status request
identify.cnf	APAM	AREP InvokeID Service type Service status Identify Data	this primitive is used to confirm an identify request
status.cnf	APAM	AREP InvokeID Service type Service status Status Data	this primitive is used to confirm a status request

8.3.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL-user and the APAM are listed in Table 8.

Table 8 – Parameters used with primitives exchanged FAL-user and APAM

Parameter	Description
AREP	This parameter contains sufficient information for local identification of the AREP to be used to convey the service.
InvokeID	This parameter identifies this invocation of the service.
Identify data	See IEC 61158-5-21:2010
Status data	See IEC 61158-5-21:2010

8.3.2 State machine

8.3.2.1 General

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

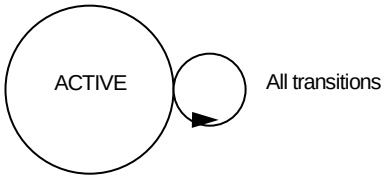

Figure 7 – State transition diagram of APAM

8.3.2.2 State tables

The APAM state machine is described in Figure 7, and in Table 9 and Table 10.

Table 9 – APAM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 10 – APAM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

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

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

R3	ACTIVE	CS_ind
		&& PDU_Type = Identify-ResponsePDU
		&& GetErrorInfo() = “success”
		=>
		Status.cnf(+){
		Data := user_data
		}
			ACTIVE

R4	ACTIVE	CS_ind
		&& PDU_Type = Identify-ResponsePDU
		&& GetErrorInfo() <> “success”
		=>
		Status.cnf(-){
		Status := GetErrorInfo()
		}
			ACTIVE

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

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

8.3.2.3 Functions

Table 11 lists the functions used by the APAM, their arguments, and their descriptions.

Table 11 – Functions used by the APAM

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

8.4 Service data object ASE protocol machine (SDOM)

8.4.1 Primitive definitions

8.4.1.1 Primitives exchanged

Table 12 shows the service primitives and their associated parameters that are exchanged between the FAL-user and the SDOM.

Table 12 – Primitives exchanged between FAL-user and SDOM

Primitive	Source	Associated parameters	Functions
Read.req	FAL-user	AREP InvokeID Object List Count Object List(n)	This primitive is used to read values of a service data object.
Write.req	FAL-user	AREP InvokeID Object List Count Object List(n) Data(n)	This primitive is used to write values of a service data object.
Read.rsp	FAL-user	AREP InvokeID Object List Count Data(n)	This primitive is used to convey requested values of a service data object.
Write.rsp	FAL-user	AREP InvokeID Service status	This primitive is used to report result of writing requested.
Read.ind	SDOM	AREP InvokeID Object List Count Object List(n)	This primitive is used to convey a read request.
Write.ind	SDOM	AREP InvokeID Object List Count Object List(n) Data(n)	This primitive is used to convey a write request.
Read.cnf	SDOM	AREP InvokeID Object List Count Data(n)	This primitive is used to convey values of data requested and result of reading.
Write.cnf	SDOM	AREP InvokeID Service status	This primitive is used to report result of writing requested.

8.4.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL-user and the SDOM are listed in Table 13.

Table 13 – Parameters used with primitives exchanged FAL-user and SDOM

Parameter	Description
AREP	This parameter contains sufficient information for local identification of the AREP to be used to convey the service.
InvokeID	This parameter identifies this invocation of the service.
Object List Count	This parameter specifies the object to which the data are to be read or written.
Object List	This parameter specifies object list.
Data	This parameter specifies object related data.

8.4.2 State machine

8.4.2.1 General

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

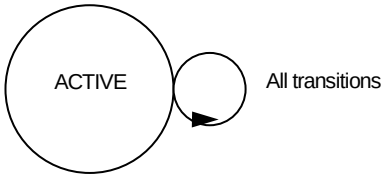

Figure 8 – State transition diagram of SDOM

8.4.2.2 State tables

The SDOM state machine is described in Figure 8, and in Table 14 and Table 15.

Table 14 – SDOM state table – Sender transitions

#	Current state	Event or condition => action	Next state

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

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

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

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

Table 15 – SDOM 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(+){
		Service status := 0
		Data := user_data
		}
			ACTIVE

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

R5	ACTIVE	CS_ind
		&& PDU_Type =Write-ResponsePDU
		&& GetErrorInfo() = “success”
		=>
		Write.cnf(+){
		Service status := 0
		Data := user_data
		}
			ACTIVE

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

8.4.2.3 Functions

Table 16 lists the functions used by the SDOM, their arguments and their descriptions.

Table 16 – Functions used by the SDOM

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

8.5 Process data object ASE protocol machine (PDOM)

8.5.1 Primitive definitions

8.5.1.1 Primitives exchanged

Table 17 shows the service primitives and their associated parameters that exchanged between the FAL-user and the PDOM.

Table 17 – Primitives exchanged between FAL-user and PDOM

Primitive	Source	Associated parameters	Functions
TB.req	FAL-user	AREP TB PDO	This primitive is used to publish values of a process data object.
COS.req	FAL-user	AREP COS PDO	This primitive is used to publish values of a process data object.
TB.ind	PDOM	AREP TB PDO	This primitive is used to report values of process data object published.
COS.ind	PDOM	AREP COS PDO	This primitive is used to report values of process data object published.

8.5.1.2 Parameters of primitives

The parameters used with the primitives exchanged between the FAL-user and the PDOM are listed in Table 18.

Table 18 – Parameters used with primitives exchanged between FAL-user and PDOM

Parameter	Description
AREP	This parameter contains sufficient information for local identification of the AREP to be used to convey the service.
TB PDO	This parameter conveys timer-based FAL-user data.
COS PDO	This parameter conveys change-of-state FAL-user data.

8.5.2 State machine

8.5.2.1 General

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

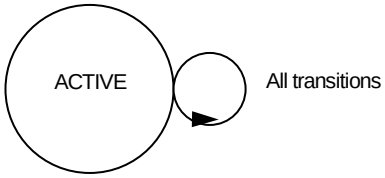

Figure 9 – State transition diagram of PDOM

8.5.2.2 State tables

The PDOM state machine is described in Figure 9, and in Table 19 and Table 20.

Table 19 – PDOM state table – Sender transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	TB.req
		=>
		SelectArep(ArepID, “MSU-AR”),
		UCS_req{
		user_data := TB-transferPDU
		}
			ACTIVE

S2	ACTIVE	COS.req
		=>
		SelectArep(ArepID, “MTU-AR”),
		UCS_req{
		user_data := COS-transferPDU
		}
			ACTIVE

Table 20 – PDOM state table – Receiver transitions

#	Current state	Event or condition => action	Next state

R1	ACTIVE	UCS_ind
		&& PDU_Type = TB-transferPDU
		=>
		TB.ind{
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

R2	ACTIVE	UCS_ind
		&& PDU_Type = COS-transferPDU
		=>
		COS.ind{
		ArepID := arep_id
		Data := user_data
		}
			ACTIVE

8.5.2.3 Functions

Table 21 lists the functions used by the SDOM, their arguments and their descriptions.

Table 21 – Functions used by the SDOM

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

9 AR protocol machine

9.1 General

This fieldbus has ARPMs for:

a) point-to-point user-triggered confirmed client/server AREP (PTC-AR);

b) multipoint network-scheduled unconfirmed publisher/subscriber AREP (MSU-AR);

c) multipoint user-triggered unconfirmed publisher/subscriber AREP (MTU-AR).

9.2 Point-to-point user-triggered confirmed client/server AREP (PTC-AR) ARPM

9.2.1 PTC-AR Primitive definitions

9.2.1.1 Primitives exchanged between PTC-ARPM and user

Table 22 and Table 23 list the primitives exchanged between the ARPM and the user.

Table 22 – Primitives issued by user to PTC-ARPM

Primitive name	Source	Associated parameters	Functions
CS_req	FSPM	Destination_dlsap_address InvokeID Service type User_data	This is an FAL internal primitive used to convey a Confirmed Send request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	Destination_dlsap_address InvokeID Service type User_data	This is an FAL internal primitive used to convey a Confirmed Send response primitive from the FSPM to the ARPM.

Table 23 – Primitives issued by PTC-ARPM to user

Primitive name	Source	Associated parameters	Functions
CS_ind	ARPM	Source_dlsap_address InvokeID Service type User_data	This is an FAL internal primitive used to convey a Confirmed Send indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	Source_dlsap_address InvokeID Service type User_data	This is an FAL internal primitive used to convey a Confirmed Send confirmation primitive from the ARPM to the FSPM.

9.2.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-21:2010.

9.2.2 DLL mapping of PTC-AREP class

9.2.2.1 Formal model

The Formal model describes the mapping of the PTC-AREP class to the Type 21 DLL defined in IEC 61158-3-21:2010 and IEC 61158-4-21:2010. It does not redefine the data link service access point (DLSAP) attributes or data link management entity (DLME) attributes that are or will be defined in the DLL specification; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes is outside the scope of this standard.

The DLL mapping attributes and their permitted values and the DLL services used with the PTC-AR AREP class are defined in this part of IEC 61158-6-21:2010.

CLASS: **PTC-AR**
PARENT CLASS: **Point-to-point user-triggered confirmed client/server AREP**
ATTRIBUTES:
1 (m) KeyAttribute: LocalDlcepAddress
2 (m) Attribute: RemoteDlcepAddress
DLL SERVICES:
1 (m) OpsService: DL-DATA

9.2.2.2 Attributes

LocalDlcepAddress
This attribute specifies the local data link connection endpoint (DLCEP) address and to identify the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

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

9.2.2.3 DLL services

Refer to IEC 61158-3-21:2010 for DLL service descriptions.

9.2.3 PTC-ARPM state machine

9.2.3.1 PTC-ARPM states

The PTC-ARPM state machine has only one state called “ACTIVE.” See Figure 10.

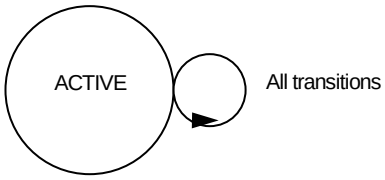

Figure 10 – State transition diagram of PTC-ARPM

9.2.3.2 PTC-ARPM state table

Table 24 and Table 25 define the state machine of the PTC-ARPM.

Table 24 – PTC-ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	CS_req
		&& Role = “Client” || “Peer”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Destination_dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “ConfirmedSend-CommandPDU,”
		fal_data := User_data)
		}
			ACTIVE

S2	ACTIVE	CS_rsp
		&& Role = “Server” || “Peer”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Destination_dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “ConfirmedSend-ResponsePDU,”
		fal_data := User_data)
		}
			ACTIVE

Table 25 – PTC-ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state

R1	ACTIVE	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = " ConfirmedSend-CommandPDU "
		&& Role = "Peer" || "Server"
		=>
		CS_ind{
		Source_dlsap_address := calling_address,
		user _data := fal_pdu
		}
			ACTIVE

R2	ACTIVE	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = " ConfirmedSend-ResponsePDU "
		&& Role = "Client" || "Peer"
		=>
		CS_cnf{
		user_data := fal_pdu
		}
			ACTIVE

9.2.3.3 Functions used by PTC-ARPM

Decoding to derive the relevant parameters for the state machine always follows receipt of an FAL-PDU_ind primitive. This is an implicit function and is not listed separately.

Table 26 defines the other function used by this state machine.

Table 26 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
Service type data additional information		DLSDU	
Function			
Builds an FAL-PDU out of the parameters given as input variables.			

9.3 Multipoint network-scheduled unconfirmed publisher/subscriber AREP (MSU-AR) ARPM

9.3.1 MSU-AR primitive definitions

9.3.1.1 Primitives exchanged between MSU-ARPM and user

Table 27 and Table 28 list the primitives exchanged between the ARPM and the user.

Table 27 – Primitives issued by user to ARPM

Primitive name	Source	Associated parameters	Functions
UCS_req	FSPM	Remote_dlsap_address User_data	This is an FAL internal primitive used to convey an Unconfirmed Send request primitive from the FSPM to the ARPM.

Table 28 – Primitives issued by ARPM to user

Primitive name	Source	Associated parameters	Functions
UCS_ind	ARPM	Remote_dlsap_address User_data	This is an FAL internal primitive used to convey an Unconfirmed Send indication primitive from the ARPM to the FSPM.

9.3.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-21:2010.

9.3.2 DLL mapping of MSU-AR class

9.3.2.1 Formal model

The Formal model describes the mapping of the MSU-AR AREP class to the Type 21 DLL defined in IEC 61158-3-21:2010 and IEC 61158-4-21:2010. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the DLL specification; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes is outside the scope of this standard.

The DLL mapping attributes with their permitted values and the DLL services used with the MTU-AR AREP class are defined in this part of IEC 61158-6-21:2010.

CLASS:			**MSU-AR**
PARENT CLASS:			**Multipoint network-scheduled unconfirmed publisher/subscriber AREP**
ATTRIBUTES:			
1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
3	(m)	Attribute:	Role (Publisher, Subscriber)
DLL SERVICES:			
1	(m)	OpsService:	DL-DATA

9.3.2.2 Attributes

LocalDlcepAddress
This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the DLCEP-address parameter of the DLL.

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

Role
This attribute specifies the role of this AREP. A value of “Publisher” indicates that this AREP is used as a publisher. The value of “Subscriber” indicates that this AREP is used as a subscriber.

9.3.2.3 DLL services

Refer to IEC 61158-3-21:2010 for DLL service descriptions.

9.3.3 MSU-ARPM state machine

9.3.3.1 MSU-ARPM states

The MSU-ARPM state machine has only one state called “ACTIVE.” See Figure 11.

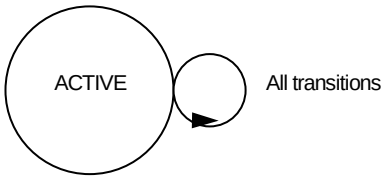

Figure 11 – State transition diagram of MSU-ARPM

9.3.3.2 MSU-ARPM state table

Table 29 and Table 30 define the state machine of the MSU-ARPM.

Table 29 – MSU-ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	UCS_req
		&& Role = “Publisher”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Remote_dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “UnconfirmedSend-CommandPDU,”
		fal_data := user_data)
		}
			ACTIVE

Table 30 – MSU-ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state

R1	ACTIVE	FAL-PDU_ind
		&& FAL_Pdu_Type (fal_pdu) = “UnconfirmedSend-CommandPDU “
		&& Role = “Subscriber”
		=>
		UCS_ind{
		remote_dlsap_address := calling_address,
		user_data := fal_pdu
		}
			ACTIVE

9.3.3.3 Functions used by MSU-ARPM

Decoding to derive the relevant parameters for the state machine always follows receipt of an FAL-PDU_ind primitive. This is an implicit function and is not listed separately.

Table 31 defines the other function used by this state machine.

Table 31 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
Service type data additional information		DLSDU	
Function			
Builds an FAL-PDU out of the parameters given as input variables.			

9.4 Multipoint user-triggered unconfirmed publisher/subscriber AREP (MTU-AR) ARPM

9.4.1 MTU-AR primitive definitions

9.4.1.1 Primitives exchanged between MTU-ARPM and user

Table 32 and Table 33 list the primitives exchanged between the ARPM and the user.

Table 32 – Primitives issued by user to ARPM

Primitive name	Source	Associated parameters	Functions
UCS_req	FSPM	Remote_dlsap_address User_data	This is an FAL internal primitive used to convey an Unconfirmed Send request primitive from the FSPM to the ARPM.

Table 33 – Primitives issued by ARPM to user

Primitive name	Source	Associated parameters	Functions
UCS_ind	ARPM	Remote_dlsap_address User_data	This is an FAL internal primitive used to convey an Unconfirmed Send indication primitive from the ARPM to the FSPM.

9.4.1.2 Parameters of primitives

The parameters of the primitives are described in IEC 61158-5-21:2010.

9.4.2 DLL mapping of MTU-AR class

9.4.2.1 Formal model

The Formal model describes mapping of the MSU-AR AREP class to the Type 21 DLL defined in IEC 61158-3-21:2010 and IEC 61158-4-21:2010. It does not redefine the DLSAP attributes or the DLME attributes that are or will be defined in the DLL specification; rather, it defines how they are used by this AR class.

NOTE A means to configure and monitor the values of these attributes is outside the scope of this standard.

This part of IEC 61158-6-21:2010 defines the DLL mapping attributes with their permitted values, and the DLL services used with the MSU-AR AREP class.

CLASS: **MTU-AR**
PARENT CLASS: **Multipoint user-triggered unconfirmed publisher/subscriber AREP**
ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
3	(m)	Attribute:	Role (Publisher, Subscriber)

DLL SERVICES:

1	(m)	OpsService:	DL-DATA

9.4.2.2 Attributes

LocalDlcepAddress
This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

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

Role
This attribute specifies the role of this AREP. The value of “Publisher” indicates that this AREP is used as a publisher. The value of “Subscriber” indicates that this AREP is used as a subscriber.

9.4.2.3 DLL services

Refer to IEC 61158-3-21:2010 for DLL service descriptions.

9.4.3 MTU-ARPM state machine

9.4.3.1 MTU-ARPM states

The MTU-ARPM state machine has only one state called “ACTIVE.” See Figure 12.

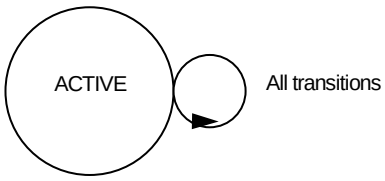

Figure 12 – State transition diagram of MTU-ARPM

9.4.3.2 MTU-ARPM state table

Table 34 and Table 35 define the state machine of the MTU-ARPM.

Table 34 – MTU-ARPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S2	ACTIVE	UCS_req
		&& Role = “Publisher”
		=>
		FAL-PDU_req {
		dlsap_id := DLSAP_ID,
		called_address := Remote_dlsap_address,
		dlsdu := BuildFAL-PDU (
		fal_pdu_name := “UnconfirmedSend-CommandPDU,”
		fal_data := user_data)
		}
			ACTIVE

Table 35 – MTU-ARPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state

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

9.4.3.3 Functions used by MTU-ARPM

Decoding to derive the relevant parameters for the state machine always follows receipt of an FAL-PDU_ind primitive. This is an implicit function and is not listed separately.

Table 36 defines the other function used by this state machine.

Table 36 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
Service type data additional information		DLSDU	
Function			
Builds an FAL-PDU out of the parameters given as input variables.			

10 DLL mapping protocol machine

10.1 Primitive definitions

10.1.1 Primitives exchanged between DMPM and ARPM

Table 37 and Table 38 list the primitives exchanged between DMPM and ARPM.

Table 37 – Primitives issued by ARPM to DMPM

Primitive name	Source	Associated parameters	Functions
FAL-PDU_req	ARPM	dmpm-service-name arep-id local-dlcep-identifier reason 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 DLL parameters that are referenced there.

Table 38 – Primitives issued by DMPM to ARPM

Primitive name	Source	Associated parameters	Functions
FAL-PDU_ind	DMPM	DLSDU	This primitive is used to pass an FAL-PDU received as a DLL service data unit to a designated ARPM.

10.1.2 Parameters of ARPM/DMPM primitives

The DLSDU parameter contains the data of the application process and all relevant information for the state machine. The DMPM state machine is able to extract this information.

10.1.3 Primitives exchanged between DLL and DMPM

Table 39 and Table 40 list the primitives exchanged between the DLL and the DMPM.

Table 39 – Primitives issued by DMPM to DLL

Primitive name	Source	Associated parameters
DL-DATA.req	DMPM	dl_dls_user_data

Table 40 – Primitives issued by DLL to DMPM

Primitive name	Source	Associated parameters
DL-DATA.ind	DLL	dl_dls_user_data

10.1.4 Parameters of DMPM/DLL primitives

The parameters used with the primitives exchanged between the DMPM and the DLL are defined in the DLL Service definition (see IEC 61158-3-21:2010). They are prefixed by “dl_” to indicate that they are used by the FAL.

10.2 DMPM state machine

10.2.1 DMPM states

The DMPM state machine has only one state called “ACTIVE,” see Figure 13.

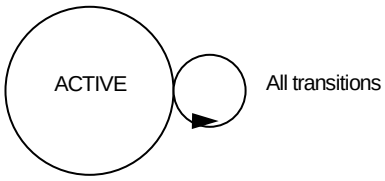

Figure 13 – State transition diagram of DMPM

10.2.2 DMPM state table

Table 41 and Table 42 define the DMPM state machine.

Table 41 – DMPM state table – sender transactions

#	Current state	Event or condition ⇒ action	Next state

S1	ACTIVE	FAL-PDU_req
		⇒
		DL-DATA.req {
		dl_dls_user_data := DLSDU
		}
			ACTIVE

Table 42 – DMPM state table – receiver transactions

#	Current state	Event or condition ⇒ action	Next state
R1	ACTIVE	DL-DATA.ind ⇒ FAL_PDU_ind	ACTIVE

10.2.3 Functions used by DMPM

Decoding to derive relevant parameters for the state machine always follows receipt of a DL-DATA.ind or a DL-DATA.ind primitive. This is an implicit function and is not listed separately.

Bibliography

IEC/TR 61158-1:2010[2], *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61588, *Precision clock synchronization protocol for networked measurement and control systems*

IEC 61784-2:2010[2], *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model: Naming and addressing*

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

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

_____

2 To be published.

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

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