IEC 61158-6-21

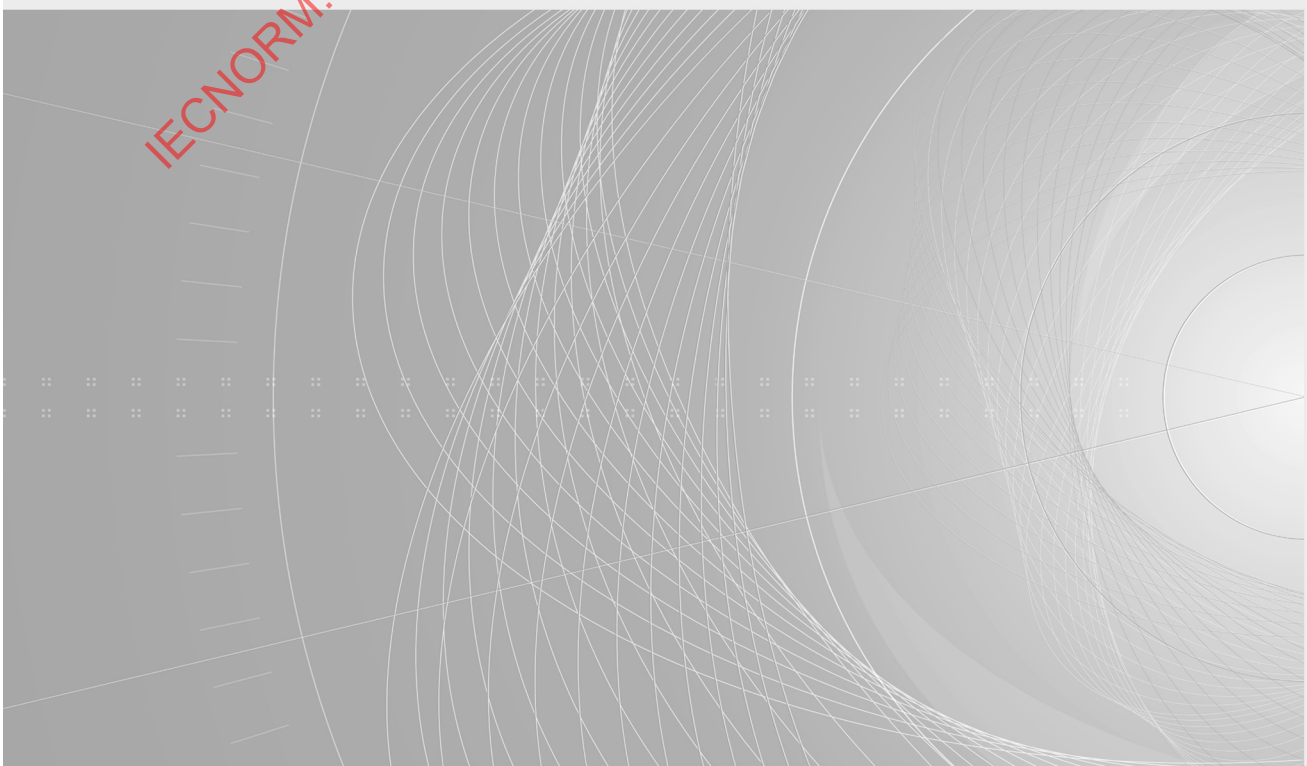
Edition 2.0 2019-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-21: Spécification du protocole de la couche application – Eléments de Type 21



IEC 61158-6-21:2019-06(en-fr)

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

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

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

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

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 It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and once a month by email.

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

Electropedia - www.electropedia.org
The world's leading online dictionary on electrotechnology, containing more than 22 000 terminological entries in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

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

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

Recherche de publications IEC - webstore.iec.ch/advsearchform
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et une fois par mois par email.

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

Electropedia - www.electropedia.org
Le premier dictionnaire d'électrotechnologie en ligne au monde, avec plus de 22 000 articles terminologiques en anglais et en français, ainsi que les termes équivalents dans 16 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
67 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

IEC 61158-6-21

Edition 2.0 2019-06

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 6-21: Spécification du protocole de la couche application – Eléments de Type 21

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-8322-9120-7

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.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in IEC 61784-1 and IEC 61784-2.

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

This second edition cancels and replaces the first edition published in 2010. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

- added WriteAndRead service;
- miscellaneous editorial corrections.

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

FDIS	Report on voting
65C/948/FDIS	65C/956/RVD

Full information on the voting for the approval of this International 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.

INTRODUCTION

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

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower layer. The primary aim of this document 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 document is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this document 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 part of IEC 61158 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the three-layer fieldbus reference model described in IEC 61158-1.

This International 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 document 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 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 document 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 document is to:

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

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

This document 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 document 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 document 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 documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61158-3-21:2019, *Industrial communication networks – Fieldbus specifications – Part 3-21: Data-link layer service definition – Type 21 elements*

IEC 61158-4-21:2019, *Industrial communication networks – Fieldbus specifications – Part 4-21: Data-link layer protocol specification – Type 21 elements*

IEC 61158-5-21:2019, *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/IEEE 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet*

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

ISO/IEC 9899, *Information technology – Programming Languages – C*

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

3 Terms, definitions, symbols, abbreviations and conventions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at http://www.electropedia.org/
- ISO Online browsing platform: available at http://www.iso.org/obp

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

Note 1 to entry: 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.

3.2.5
application process object class
class of application process objects defined in terms of the set of their network-accessible attributes and services

3.2.6
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 1 to entry: This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities.

3.2.7
application relationship application service element
application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.2.8
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 1 to entry: Each application process involved in the application relationship maintains its own application relationship endpoint.

3.2.9
attribute
description of an externally visible characteristic or feature of an object

Note 1 to entry: 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.

3.2.10
behavior
indication of how an object responds to particular events

3.2.11
channel
single physical or logical link of an input or output application object of a server to the process

3.2.12
class
set of objects, all of which represent the same type of system component

Note 1 to entry: 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.

3.2.13
class attributes
attribute shared by all objects within the same class

3.2.14
class code
unique identifier assigned to each object class

3.2.15
class-specific service
service defined by a particular object class to perform a required function that is not performed by a common service

Note 1 to entry: A class-specific object is unique to the object class that defines it.

3.2.16
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

3.2.17
consume
act of receiving data from a producer

3.2.18
consumer
node or sink that receives data from a producer

3.2.19
consuming application
application that consumes data

3.2.20
conveyance path
unidirectional flow of APDUs across an application relationship

3.2.21
cyclic
repetitive in a regular manner

3.2.22
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.2.23
device
physical hardware connected to the link

Note 1 to entry: A device may contain more than one node.

3.2.24
device profile
collection of device-dependent information and functionality providing consistency between similar devices of the same device type

3.2.25
diagnostic information
all data available at the server for maintenance purposes

3.2.26
end node
producing or consuming node

3.2.27
endpoint
one of the communicating entities involved in a connection

3.2.28
error
discrepancy between a computed, observed, or measured value or condition and the specified or theoretically correct value or condition

3.2.29
error class
general grouping for related error definitions and corresponding error codes

3.2.30
error code
identification of a specific type of error within an error class

3.2.31
event
instance of a change of conditions

3.2.32
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 1 to entry: In a fieldbus system, only one complete element can be transferred as a result of one service invocation.

3.2.33
frame
simplified synonym for data link protocol data unit (DLPDU)

3.2.34
group
<general> general term for a collection of objects

3.2.35
group
<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 1 to entry: 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 1 to entry: 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 1 to entry: 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 1 to entry: 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

Note 1 to entry: Two types of publishing managers are defined by this document, 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 document uses the descriptive conventions given in ISO/IEC 10731.

This document uses the descriptive conventions given in IEC 61158-5-21 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.

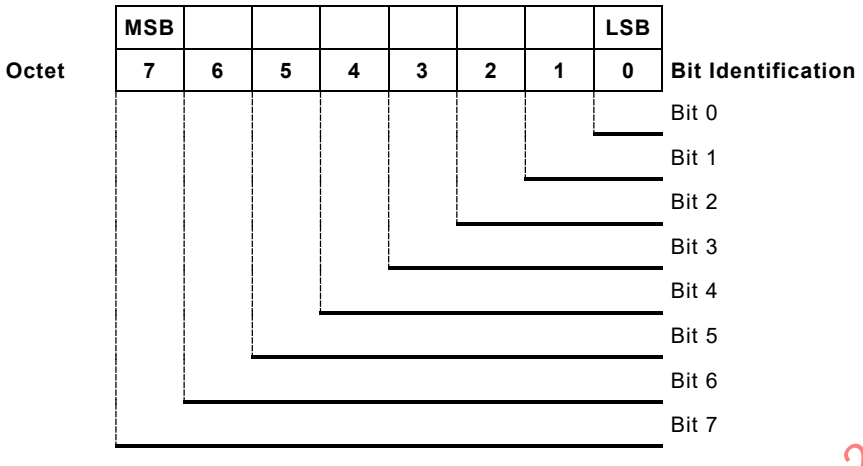


Figure 1 – Common structure of specific fields

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

3.4.5 Conventions for APDU transfer syntax definitions

This document 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,
      WriteAndRead-Request            [4]   IMPLICIT      WriteAndRead-RequestPDU,
      WriteAndReadMultiple-Request    [5]   IMPLICIT      WriteAndReadMultiple-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,
      WriteAndRead-Response           [4]   IMPLICIT      WriteAndRead-ResponsePDU,
      WriteAndReadMultiple-Response   [5]   IMPLICIT      WriteAndReadMultiple-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,
}
```

4.3.4.2 Error class

```
   ErrorClass::= CHOICE {
      noError                         [0]   IMPLICIT      Integer16 {
                                                            normal                          (0),
                                                            Other                           (1)
      }
      Definition                      [1]   IMPLICIT      Integer16 {
                                                            other                           (0),
                                                            object-undefined                (1),
      }
      Resource                        [2]   IMPLICIT      Integer16 {
                                                            other                           (0),
                                                            memory-unavailable              (1)
      }
      Service                         [3]   IMPLICIT      Integer16 {
                                                            other                           (0),
                                                            pdu-size                        (1),
                                                            illegal-parameter               (2)
      }
      Access                          [4]   IMPLICIT      Integer16 {
                                                            other                           (0),
                                                            object-access-denied            (1),
                                                            invalid-address                 (2),
                                                            object-access-unsupported       (3),
                                                            object-non-existent             (4),
      }
      Other                           [5]   IMPLICIT      Integer16 {
                                                            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
0x00	No Error	No Error
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 {
  Service status
  Status code
  DL–entity identifier
  MAC address
  Port information
  Protocol version
  Reserved8
  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 {
  Service status
  Status code
  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 Service status

Service status::= Unsigned16 — Provides information on the result of service execution.

4.4.1.6 Status code

Status code::= Unsigned16 — See 4.3.4.3.

4.4.1.7 DL–entity identifier

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

4.4.1.8 MAC address

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

4.4.1.9 Port information

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

4.4.1.10 Protocol version

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

4.4.1.11 Reserved8

Reserved8::= Unsigned8 — Reserved

4.4.1.12 Device type

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

4.4.1.13 Device description

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

4.4.1.14 Hardware-Version

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

4.4.1.15 Serial Number

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

4.4.1.16 Software-Version

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

4.4.1.17 Software-Date

Software-Date::= Unsigned16 — Contains the software date of the device.

4.4.1.18 Vendor ID

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

4.4.1.19 Product Code

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

4.4.1.20 Device flags

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

4.4.1.21 Device state

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

4.4.1.22 tx_cnt_normal

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

4.4.1.23 tx_cnt_all

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

4.4.1.24 rx_cnt_normal

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

4.4.1.25 rx_cnt_all

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

4.4.1.26 relay_cnt_normal

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

4.4.1.27 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
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   Object identifier (m)
   Data type
   offset
   length
   ...                                  — (further read requests)
}
Read-ResponsePDU::= SEQUENCE {
   Service status
   Status code
   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
   Reserved16
   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
}
```

4.4.2.3 WriteAndRead service PDUs

```
WriteAndRead-RequestPDU::= SEQUENCE {
   Object List Count  (Read)
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   Object identifier (m)
   Data type
   offset
   length
   …                                    — (further read requests)
   Object List Count  (Write)
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   data
   Object identifier (m)
   Data type
   offset
   length
   data
   …                                    — (further write requests)
}
WriteAndRead-ResponsePDU::= SEQUENCE {
   Service status (Write)
   Status code
   Service status (Read)
   Status code
   Object List Count
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   data
   Object identifier (m)
   Data type
   offset
   length
   data
   …                                    — (further read requests)
}
```

4.4.2.4 WriteAndReadMultiple service PDUs

```
WriteAndReadMultiple-RequestPDU::= SEQUENCE {
   Device List Count
   Reserved16
   Device Address
   Device Offset
   Object List Count  (Read)
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   Object identifier (m)
   Data type
   offset
   length
   ...                                  — (further read requests)
   Object List Count  (Write)
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   data
   Object identifier (m)
   Data type
   offset
   length
   data
   ...                                  — (further write requests)
}
WriteAndReadMultiple-ResponsePDU::= SEQUENCE {
   Device List Count
   Reserved16
   Device Address
   Device Offset
   Service status (Write)
   Status code
   Service status (Read)
   Status code
   Object List Count
   Reserved16
   Object identifier (k)
   Data type
   offset
   length
   data
   Object identifier (m)
   Data type
   offset
   length
   data
   ...                                  — (further read requests)
}
```

4.4.2.5 Service status

Service status::= Unsigned16 — Provides information on the result of service execution.

4.4.2.6 Status code

Status code::= Unsigned16 — See 4.3.4.3.

4.4.2.7 Object List Count

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

4.4.2.8 Reserved16

Reserved16::= Unsigned16 — Reserved.

4.4.2.9 Object identifier

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

4.4.2.10 Data type

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

4.4.2.11 Offset

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

4.4.2.12 Length

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

4.4.2.13 data

data::= Any — Application-dependent type and length;
this field shall assure unsigned32 alignment;
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 document. 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 document;

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 document 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.

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 $b = b_0 \ldots b_{n-1}$ 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/IEEE 8802-3):

$b_0, b_1, \ldots, b_7, b_8, \ldots, b_{15}, \ldots$

EXAMPLE

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

The bit sequence $b = b_0 \ldots b_9 = 0011\ 1000\ 01_b$ 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 the value

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

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 the value

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

and, performing two's complement arithmetic,

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

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 in 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 the value (finite non-zero number)

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

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/IEC 646 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 document.

NOTE The state machines specified in this document 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.

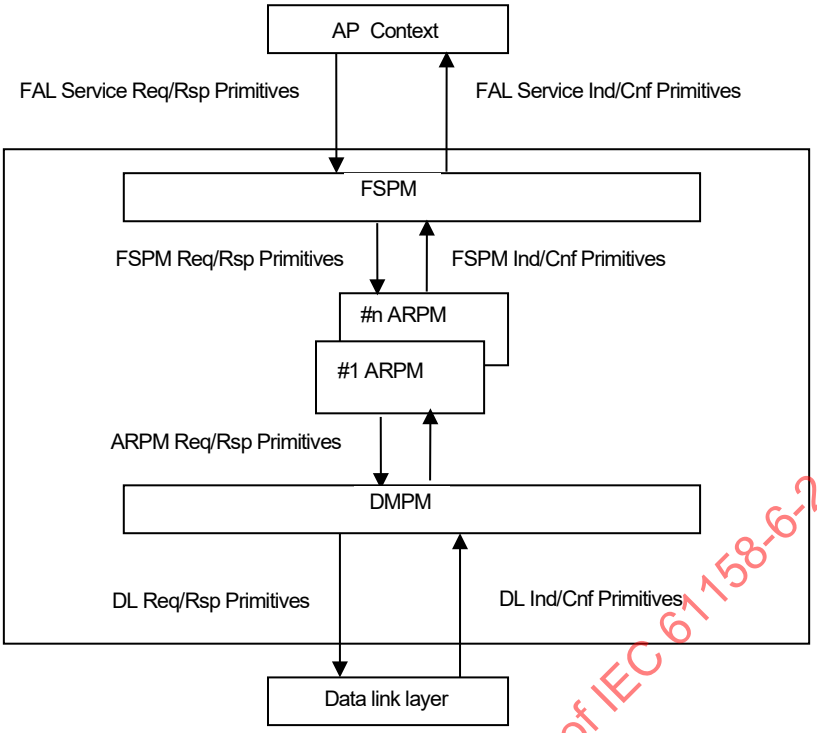


Figure 6 – Primitives exchanged between protocol machines

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.

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
Status data	See IEC 61158-5-21

8.3.2 State machine

8.3.2.1 General

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

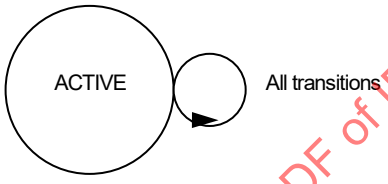


Figure 7 – State transition diagram of APAM

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.

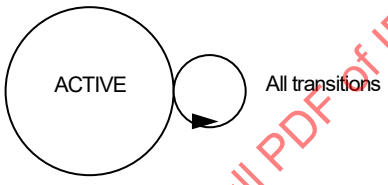


Figure 8 – State transition diagram of SDOM

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.

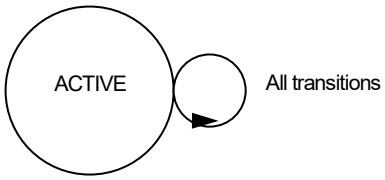


Figure 9 – State transition diagram of PDOM

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.

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 and IEC 61158-4-21. 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 document.

The DLL mapping attributes and their permitted values and the DLL services used with the PTC-AR AREP class are defined in this subclause 9.2.2.

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 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.

9.2.2.3 DLL services

Refer to IEC 61158-3-21 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.

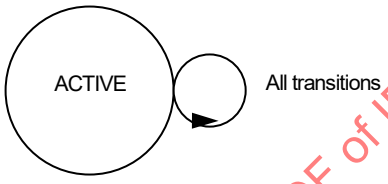


Figure 10 – State transition diagram of PTC-ARPM

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.

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 and IEC 61158-4-21. 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 document.

The DLL mapping attributes with their permitted values and the DLL services used with the MTU-AR AREP class are defined in this subclause 9.3.2.

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 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.

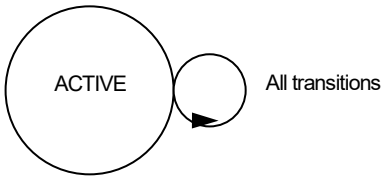


Figure 11 – State transition diagram of MSU-ARPM

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.

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 and IEC 61158-4-21. 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 document.

This subclause 9.4.2 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 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.

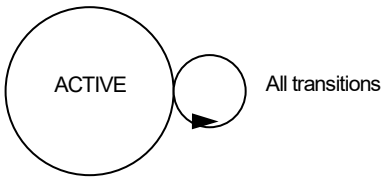


Figure 12 – State transition diagram of MTU-ARPM

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). 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.

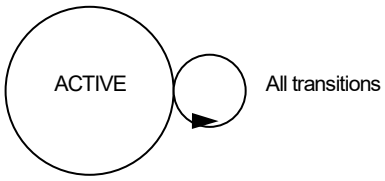


Figure 13 – State transition diagram of DMPM

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 61158-1:2019, *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 61784-2:2019, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC/IEEE 8802-3*

ISO/IEC 646, *Information technology – ISO 7-bit coded character set for information interchange*

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)*

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-21: Spécification du protocole de la couche application – Eléments de Type 21

AVANT-PROPOS

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

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

NOTE Les combinaisons de types de protocoles sont spécifiées dans les normes IEC 61784-1 et IEC 61784-2.

La Norme internationale IEC 61158-6-21 a été établie par le sous-comité 65C: Réseaux industriels, du comité d'études 65 de l'IEC: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition parue en 2010. Cette édition constitue une révision technique.

La présente édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

- ajout du service WriteAndRead;
- corrections rédactionnelles diverses.

La présente version bilingue (2020-12) correspond à la version anglaise monolingue publiée en 2019-06.

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

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

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

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

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

INTRODUCTION

La présente partie de l'IEC 61158 appartient à une série produite pour faciliter l'interconnexion des composants d'un système d'automation. Elle est liée à d'autres normes de la série telle que définie par le modèle de référence des bus de terrain "à trois couches" décrit dans l'IEC 61158-1.

Le protocole application fournit le service application en utilisant les services disponibles de la liaison de données ou autre couche immédiatement inférieure. Le principal objectif du présent document est de définir un ensemble de règles de communication, exprimées en termes de procédures que doivent suivre les entités d'application (Application Entity, AE) homologues au moment de la communication. Ces règles pour la communication visent à fournir une base solide pour le développement et de servir une diversité de besoins:

- guider les implémenteurs et les concepteurs;
- pour une utilisation dans les essais et achats d'équipements;
- comme partie intégrante d'un accord pour l'admission de systèmes dans l'environnement de systèmes ouverts;
- comme affinement pour la compréhension de communications prioritaires au sein de l'OSI (Open Systems Interconnexion, c'est-à-dire Interconnexion des systèmes ouverts).

Cette norme traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automatisation. L'utilisation conjointe du présent document avec d'autres normes entrant dans les modèles de référence OSI ou de bus de terrain permet à des systèmes de fonctionner ensemble dans toute combinaison, ce qu'ils ne pourraient pas faire autrement.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-21: Spécification du protocole de la couche application – Eléments de Type 21

1 Domaine d'application

1.1 Généralités

La présente partie de l'IEC 61158 appartient à une série élaborée pour faciliter l'interconnexion des composants des systèmes d'automatisation. Elle est liée à d'autres normes de la série telle que définie par le modèle de référence des bus de terrain "à trois couches" décrit dans l'IEC 61158-1.

La présente Norme internationale contient des éléments spécifiques au protocole de communication de Type 21.

1.2 Vue d'ensemble

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

Le présent document fournit les éléments communs pour les communications à temps critique et à temps non critique entre des application dans un environnement d'automatisation ainsi que des éléments spécifiques au protocole de Type 21. Le terme "prioritaire" / "à temps critique" est utilisé pour traduire la présence d'une fenêtre temporelle, à l'intérieur de laquelle une ou plusieurs actions spécifiées doivent être réalisées selon un niveau défini de certitude. Si certaines actions spécifiées ne sont pas terminées dans les limites du délai requis, cela risque d'entraîner la défaillance des applications qui demandent ces actions, avec un risque associé pour l'équipement, la centrale et éventuellement pour les personnes.

Le présent document définit les interactions entre applications distantes. Il définit également le comportement visible de l'extérieur fourni par la couche application de Type 21 en termes:

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

b) de syntaxe de transfert définissant les règles de codage qui s'appliquent aux APDU;

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

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

Le présent document a pour objet de:

a) définir la représentation filaire des primitives de service spécifiées dans l'IEC 61158-5-21;

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

Le présent document définit le protocole de la couche application de Type 21, en conformité avec le modèle de référence de base OSI (ISO/IEC 7498) et la structure de la couche application OSI (ISO/IEC 9545).

1.3 Spécifications

Le présent document a pour objectif principal de spécifier la syntaxe et le comportement du protocole de la couche application qui véhicule les services de la couche application de type 21.

Un objectif secondaire consiste à fournir des trajets de migration à partir de protocoles industriels de communication préexistants.

1.4 Conformité

Le présent document ne limite pas les mises en œuvre individuelles ou les produits individuels et il ne contraint pas non plus les mises en œuvre d'entités de la couche application au sein des systèmes d'automatisation industriels. La conformité est obtenue par la mise en œuvre de la présente spécification du protocole de couche application.

2 Références normatives

Les documents suivants sont cités dans le texte de sorte qu'ils constituent, pour tout ou partie de leur contenu, des exigences du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

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

IEC 61158-3-21:2019, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 3-21: Définition des services de couche liaison de données – Eléments de Type 21*

IEC 61158-4-21:2019, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 4-21: Spécification du protocole de la couche de liaison de données – Eléments de Type 21*

IEC 61158-5-21:2019, *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-21: Définition des services de la couche application – Eléments de type 21*

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

ISO/IEC/IEEE 88023, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet* (disponible en anglais seulement)

ISO/IEC 8822, *Technologies de l'information – Interconnexion de systèmes ouverts –Définition du service de présentation*

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

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

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

ISO/IEC 9899, *Information technology – Programming Languages – C* (disponible en anglais seulement)

IEEE 754-2008, *IEEE Standard forBinary Floating-Point Arithmetic* (disponible en anglais seulement)

3 Termes, définitions, symboles, abréviations et conventions

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

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse http://www.electropedia.org/
- ISO Online browsing platform: disponible à l'adresse http://www.iso.org/obp

3.1 Termes et définitions provenant d'autres normes ISO/IEC

3.1.1 Termes de l'ISO/IEC 7498-1

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

a) entité d'application
b) processus d'application
c) unité de données de protocole d'application
d) élément de service d'application
e) invocation d'entité d'application
f) invocation de processus d'application
g) transaction d'application
h) système ouvert réel
i) syntaxe de transfert

3.1.2 Termes de l'ISO/IEC 8822

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

a) syntaxe abstraite
b) contexte de présentation

3.1.3 Termes de l'ISO/IEC 8824-1

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

a) identificateur d'objet
b) type

3.1.4 Termes de l'ISO/IEC 9545

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

a) association d'application
b) contexte d'application
c) nom de contexte d'application
d) invocation d'entité d'application

e) type d'entité d'application

f) invocation de processus d'application

g) type de processus d'application

h) élément de service application

i) élément de service de commande d'application

3.2 Autres termes et définitions

3.2.1
application
fonction ou structure de données pour laquelle les données sont consommées ou produites

3.2.2
objets d'application
classes d'objets multiples qui gèrent et assurent l'échange de messages pendant le mode exécution à travers le réseau et à l'intérieur du dispositif de réseau

3.2.3
processus application
partie d'une application répartie sur un réseau, qui est située sur un dispositif et traitée sans ambiguïté

3.2.4
objet de processus d'application
composant d'un processus d'application qui est identifiable et accessible par la relation d'application de la FAL

Note 1 à l'article: Les définitions d'objet de processus d'application se composent d'un jeu de valeurs pour les attributs de leur classe (voir la définition de "classe d'objets de processus d'application"). L'accès aux définitions d'objet de processus d'application peut se faire à distance à l'aide des services de l'ASE de gestion d'objet FAL. Les services de Gestion d'objet de la FAL peuvent être utilisés pour charger ou mettre à jour des définitions d'objet, pour lire des définitions d'objet, et pour créer et supprimer de manière dynamique des objets d'application et leurs définitions correspondantes.

3.2.5
classe d'objet de processus d'application
classe d'objets de processus d'application définie par l'ensemble de leurs attributs et services accessibles en réseau

3.2.6
relation d'application
association coopérative entre au moins deux invocations d'entité d'application, destinée à l'échange d'informations et à la coordination de leur association fonctionnelle

Note 1 à l'article: Cette relation est activée soit par l'échange d'unités de données de protocole d'application (application-protocol-data-unit), soit à la suite d'activités de préconfiguration.

3.2.7
élément de service d'application de relation d'applications
élément de service d'application qui fournit le moyen exclusif pour établir et pour terminer toutes les relations d'application

3.2.8
point d'extrémité de relation d'application
contexte et comportement d'une relation d'application, tels qu'ils sont vus et maintenus par l'un des processus d'application impliqués dans la relation d'application

Note 1 à l'article: Chaque processus d'application impliqué dans la relation d'applications maintient son propre point d'extrémité de relation d'application.

3.2.9
attribut
description d'une caractéristique ou d'une particularité, visible de l'extérieur, d'un objet

Note 1 à l'article: Les attributs d'un objet contiennent des informations relatives aux portions variables d'un objet. Ils servent habituellement à fournir des informations de statut ou à régir le fonctionnement d'un objet. Les attributs peuvent également avoir un impact sur le comportement d'un objet. Les attributs se répartissent en attributs de classe et en attributs d'instance.

3.2.10
comportement
indication de la manière dont un objet réagit à des événements particuliers

3.2.11
canal
liaison physique ou logique unique d'un objet d'application d'entrée ou de sortie d'un serveur avec le processus

3.2.12
classe
ensemble d'objets, qui représentent tous le même type de composant système

Note 1 à l'article: Une classe est une généralisation d'un objet, un modèle qui permet de définir des variables et des méthodes. Tous les objets d'une classe sont de forme et de comportement identiques, mais ils contiennent généralement des données différentes dans leurs attributs.

3.2.13
attribut de classe
attribut partagé par tous les objets au sein de la même classe

3.2.14
code de classe
identificateur unique affecté à chaque classe d'objet

3.2.15
service spécifique à une classe
service défini par une classe d'objet particulière pour accomplir une fonction requise qui n'est pas accomplie par un service commun

Note 1 à l'article: Tout objet spécifique à une classe appartient exclusivement à la classe d'objets qui le définit.

3.2.16
client
a) objet qui utilise les services d'un autre objet (serveur) pour accomplir une tâche

b) initiateur d'un message auquel un serveur réagit

3.2.17
consommer
acte consistant à recevoir des données provenant d'un producteur

3.2.18
consommateur
nœud ou puits qui reçoit des données d'un producteur

3.2.19
application consommatrice
application qui consomme des données

3.2.20
chemin de transmission
flux unidirectionnel d'APDU sur l'ensemble d'une relation d'application

3.2.21
cyclique
qui se reproduit de manière régulière et répétitive

3.2.22
cohérence des données
moyen pour assurer une transmission et un accès cohérents de l'objet de données d'entrée ou de sortie entre un client et un serveur et à l'intérieur de ceux-ci

3.2.23
appareil
matériel physique connecté à la liaison

Note 1 à l'article: Un appareil peut contenir plus d'un nœud.

3.2.24
profil d'appareil
collection d'informations et de fonctionnalités qui dépendent du dispositif, et assurent une harmonisation entre dispositifs similaires de même type

3.2.25
informations de diagnostic
toutes les données disponibles sur le serveur aux fins de maintenance

3.2.26
nœud d'extrémité
nœud producteur ou consommateur

3.2.27
point d'extrémité
l'une des entités communicantes impliquées dans une connexion

3.2.28
erreur
discordance entre une valeur ou un état calculé(e), observé(e) ou mesuré(e) et la valeur ou l'état spécifié(e) ou théoriquement correct(e)

3.2.29
classe d'erreur
groupement général des définitions d'erreur connexes et des codes d'erreur correspondants

3.2.30
code d'erreur
identification d'un type d'erreur spécifique dans une classe d'erreur

3.2.31
événement
instance d'un changement des conditions

3.2.32
variable FIFO
classe d'objets variables composée d'un jeu d'éléments de type homogène, dans laquelle le premier élément écrit est le premier élément qui peut être lu

Note 1 à l'article: Dans un système de bus de terrain, un seul élément commun peut être transféré à la suite d'une invocation de service.

3.2.33
trame
synonyme simplifié pour l'unité de données de protocole de liaison de données (DLPDU, data link protocol data unit)

3.2.34
groupe
<général> terme général pour un ensemble d'objets

3.2.35
groupe
<adressage> lors de la description d'une adresse, adresse qui identifie plus d'une entité

3.2.36
invocation
acte consistant à utiliser un service ou une autre ressource d'un processus d'application

Note 1 à l'article: Chaque invocation représente un fil de commande distinct qui peut être décrit par son contexte. Une fois le service terminé ou la ressource libérée, l'invocation cesse d'exister. Dans le cas des invocations de services, un service lancé, mais pas encore terminé est considéré comme une invocation de services en cours. Pour des invocations de service, un "Invoke ID" peut être utilisé pour identifier sans ambiguïté l'invocation de service et la différencier d'autres invocations de service en cours.

3.2.37
index
adresse d'un objet au sein d'un processus application

3.2.38
instance
occurrence physique réelle d'un objet au sein d'une classe qui identifie un objet parmi de nombreux autres objets dans la même classe d'objets

EXEMPLE "California" (La Californie) est une instance de la classe d'objets "state" (état).

Note 1 à l'article: Les termes objet, instance et instance d'objet font référence à une instance particulière.

3.2.39
attributs d'instance
attribut unique d'une instance d'objet, qui n'est pas partagé par la classe d'objets

3.2.40
instancié
objet créé dans un appareil

3.2.41
dispositif logique
classe spécifique de la FAL qui abstrait un composant logiciel ou un composant de micrologiciel comme une fonction monobloc autonome d'un dispositif d'automation

3.2.42
ID du fabricant
identification de chaque fabricant de produits par un numéro unique

3.2.43
information de gestion
informations accessibles par le réseau qui viennent à l'appui de la gestion du fonctionnement du système de bus de terrain, y compris la couche application

Note 1 à l'article: La gestion inclut des fonctions telles que la commande, la surveillance et le diagnostic.

3.2.44
réseau
ensemble de nœuds reliés par un certain type de support de communication, notamment des répétiteurs intermédiaires, ponts, routeurs et passerelles de couche inférieure

3.2.45
objet
représentation abstraite d'un composant particulier au sein d'un dispositif, habituellement un ensemble de données connexes sous la forme de variables, et de méthodes (procédures) pour effectuer des opérations sur les données en question qui ont une interface et un comportement clairement définis

3.2.46
dictionnaire d'objets
ensemble de définitions, d'attributs et de paramètres spécifiques aux communications et de données dépendant de l'application

3.2.47
service spécifique à un objet
service propre à la classe d'objets qui le définit

3.2.48
appareil physique
appareil d'automation ou autre appareil de réseau

3.2.49
connexion point à point
connexion établie précisément entre deux objets d'application

3.2.50
point d'extrémité d'AR préétabli
point d'extrémité d'AR qui est mis dans un état établi pendant la configuration des AE qui commandent ses points d'extrémité

3.2.51
données de processus
objet(s) déjà prétraité(s) et transféré(s) de façon cyclique à des fins d'informations ou de traitement ultérieur

3.2.52
produire
acte d'envoyer des données destinées à être reçues par un consommateur

3.2.53
producteur
nœud chargé de l'envoi des données

3.2.54
propriété
terme général désignant toute information descriptive concernant un objet

3.2.55
fournisseur
source d'une connexion de données

3.2.56
éditeur
rôle d'un point d'extrémité d'AR qui transmet les unités APDU sur le bus de terrain afin qu'elles soient consommées par un ou plusieurs abonnés

Note 1 à l'article: Un éditeur peut ne pas connaître l'identité des abonnés ou leur nombre.

3.2.57
gestionnaire de publication
rôle d'un point d'extrémité d'AR dans lequel il émet une ou plusieurs unités de donnée de protocole application (APDU, application protocol data unit) de demande de services confirmés vers un éditeur pour demander qu'un objet spécifié soit édité.

Note 1 à l'article: Le présent document définit deux types de gestionnaires de publication, à savoir les gestionnaires de publication en mode pull et les gestionnaires de publication en mode push, chacun de ceux-ci étant défini séparément

3.2.58
éditeur push
type d'éditeur qui publie un objet dans une APDU de demande de service non confirmée

3.2.59
gestionnaire de publication push
type de gestionnaire de publication qui demande qu'un objet spécifié soit publié au moyen d'un service non confirmé

3.2.60
souscripteur push
type de souscripteur qui reconnaît les APDU de demande de service non confirmé reçues en tant que données d'objet fournies

3.2.61
expéditeur
utilisateur de service qui envoie une primitive confirmée ou non confirmée, ou fournisseur de service qui envoie une APDU confirmée ou non confirmée

3.2.62
serveur
a) rôle d'un point d'extrémité de relation d'applications (AREP, application relationship endpoint (AREP) dans lequel il retourne au client qui a lancé la demande une APDU confirmée de réponse de service

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

3.2.63
service
opération ou fonction qu'un objet et/ou une classe d'objets exécute à la demande d'un autre objet et/ou une autre classe d'objets

3.2.64
station
hôte d'un AP, identifié par une adresse unique de point d'extrémité de connexion de liaison de données (DLCEP, data link connection endpoint)

3.2.65
souscripteur
rôle joué par un AREP consistant à recevoir des APDU produites par un fournisseur

3.2.66
récepteur
utilisateur de service qui reçoit une primitive confirmée ou non confirmée, ou fournisseur de service qui reçoit une APDU confirmée ou non confirmée

3.2.67
ressource
capacité de traitement ou d'information d'un sous-système

3.3 Abréviations et symboles

AE	Application Entity (Entité d'application)
AL	Application Layer (Couche application)
ALME	Application Layer Management Entity (Entité de gestion de couche application)
ALP	Application Layer Protocol (Protocole de couche application)
APO	Application Object (Objet d'application)
AP	Application Process (Processus d'application)
APDU	Application Protocol Data Unit (Unité de données de protocole d'application)
AR	Application Relationship (Relation d'application)
AREP	Application Relationship End Point (Point d'extrémité de relation d'application)
ASCII	American Standard Code for Information Interchange (Code normalisé américain pour l'échange d'informations)
ASE	Application Service Element (Elément de service d'application)
Cnf	Confirmation
DL-	(préfixe) Data Link- (Liaison de données)
DLCEP	Data Link Connection End Point (Point d'extrémité de connexion de liaison de données)
DLL	Data Link Layer (Couche liaison de données)
DLM	Data Link-management (Gestion de liaison de données)
DLSAP	Data Link Service Access Point (Point d'accès au service de liaison de données)
DLSDU	DL-service-data-unit (Unité de données de service de liaison de données)
DNS	Domain Name Service (Service de noms de domaine)
FAL	Fieldbus Application Layer (Couche application de bus de terrain)
Ind	Indication
Req	Request (Demande)
Rsp	Response (Réponse)

3.4 Conventions

3.4.1 Conventions générales

Le présent document utilise les conventions descriptives données dans l'ISO/IEC 10731.

Le présent document utilise les conventions descriptives données dans l'IEC 61158-5-21 pour les définitions des services de la FAL.

3.4.2 Convention de codage des bits et octets réservés

Le terme "réservé" peut être utilisé pour décrire les bits dans des octets ou des octets complets. Il convient que tous les bits ou octets qui sont réservés soient mis à zéro du côté envoi. Ils ne seront pas soumis à essai du côté réception, sauf si cela est explicitement indiqué, ou si les bits ou octets réservés sont vérifiés par un diagramme d'états.

Le terme "réservé" peut également être utilisé pour indiquer que certaines valeurs dans la plage d'un paramètre sont réservées à des extensions ultérieures. Dans ce cas, il convient de ne pas utiliser les valeurs réservées du côté envoi. Celles-ci ne doivent pas être soumises à essai du côté réception, sauf si cela est explicitement indiqué ou si les valeurs réservées sont vérifiées par un diagramme d'états.

3.4.3 Conventions de codages communs des octets de champs spécifiques

Les APDU peuvent contenir des champs spécifiques qui contiennent des informations d'une manière primitive et condensée. Ces champs doivent être codés dans l'ordre conformément à la Figure 1.

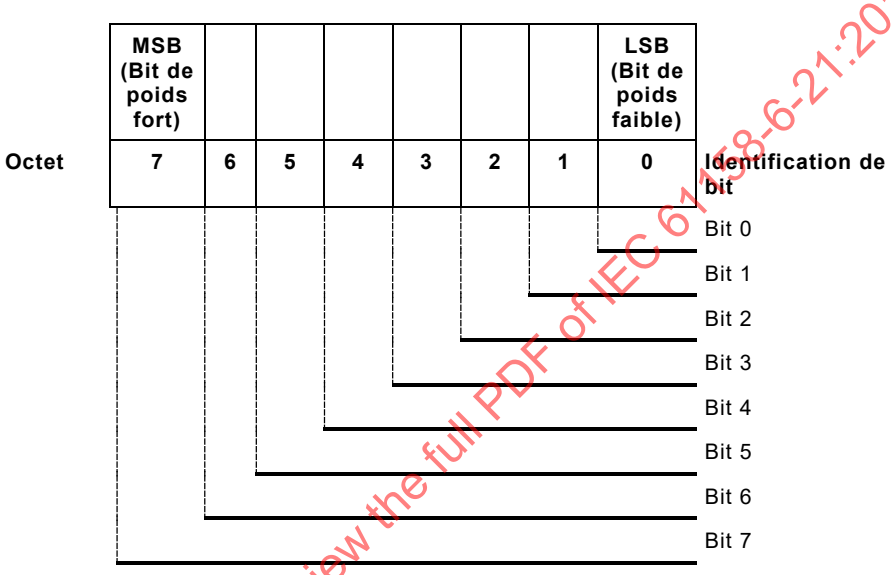


Figure 1 – Structure commune de champs particuliers

Les bits peuvent être regroupés. Chaque bit ou groupe de bits doit être adressé par son Bit Identification (identification de bit) (par exemple: Bit 0, Bit 1 à -4). La position à l'intérieur de l'octet doit être telle que représentée sur la Figure 1. Des pseudonymes peuvent être utilisés pour chaque bit ou groupe de bits ou bien ils peuvent être marqués comme étant réservés. Le regroupement de bits pris séparément doit être dans l'ordre croissant sans trous. Les valeurs d'un groupe de bits peuvent être représentées comme des valeurs binaires, décimales ou hexadécimales. Cette valeur doit être seulement valide pour les bits regroupés et peut seulement représenter l'octet tout entier si tous les 8 bits sont regroupés. Les valeurs décimales ou hexadécimales doivent être transformées en valeurs binaires afin que le bit ayant le numéro de groupe le plus élevé représente le bit de poids fort (MSB, Most Significant Bit) par rapport aux bits regroupés.

EXEMPLE Description et associations pour l'octet de champ spécifique

Bit 0: réservé.

Bits 1-3: Reason_Code. La valeur décimale 2 pour Reason_Code (code de motif) signifie "erreur générale".

Bits 4-7: Toujours mis à un.

L'octet qui est construit conformément à la description ci-dessus ressemble à ce qui suit:

(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.

Cette combinaison de bits comporte une représentation de valeur d'octet 0xf4.

3.4.4 Conventions pour les définitions de syntaxe abstraite des APDU

Le présent document utilise les conventions descriptives données dans l'ISO/IEC 8824-2 pour les définitions des APDU.

3.4.5 Conventions pour les définitions de syntaxe de transfert des APDU

Le présent document utilise les conventions descriptives données dans l'ISO/IEC 8825-1 pour les définitions de syntaxe de transfert.

3.4.6 Conventions pour les définitions de diagramme d'états d'AE

Les conventions utilisées pour les définitions de diagramme d'états d'AE sont décrites dans le Tableau 1.

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

N°	état actuel,	Événement /Condition => Action	état suivant
Nom du passage	Etat actuel dans lequel se trouve le diagramme d'états lors de ce passage d'état	Événements ou conditions qui déclenchent cette transaction d'état. Les actions qui sont entreprises lorsque les événements ou conditions ci-dessus se produisent. Elles figurent toujours au-dessous des événements ou conditions et sont toujours présentées avec un retrait.	Le prochain état qui suit les actions de cette transition est pris.

Les conventions utilisées dans les descriptions des événements, des conditions et des actions sont les suivantes:

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

xxx nom du paramètre.

Exemple:

Identifier:= Reason

signifie que la valeur du paramètre "Reason" (motif) est attribuée à un paramètre appelé "Identifier" (identificateur)

"xxx" indique une valeur fixe.

Exemple:

Identifier := "abc"

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

= Condition logique indiquant qu'une entité de gauche est égale à une entité de droite.

< Condition logique indiquant qu'une entité de gauche est inférieure à une entité de droite.

> Condition logique indiquant qu'une entité de gauche est supérieure à une entité de droite.

<> Condition logique indiquant qu'une entité de gauche n'est pas égale à une entité de droite.

&& Logique AND

|| indique le "OU" logique

La séquence d'actions et les actions en variante peuvent être exécutées en utilisant les mots réservés suivants.

for

endfor

if (si)

else

elseif (autre si)

Les exemples de description utilisant les mots réservés sont donnés ci-dessous.

Exemple 1

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

exemple 2:

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

4 Description de la syntaxe de FAL

4.1 Généralités

Cette description de la syntaxe abstraite de Type 21 utilise des formalismes similaires à ceux de l'ASN.1, bien que les règles de codage diffèrent de la présente norme.

4.2 Syntaxe abstraite des unités PDU FAL-AR

4.2.1 Définition du niveau supérieur

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

4.2.2 Service "Confirmed send" (envoi confirmé)

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

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

4.2.3 Service "unconfirmed send" (envoi non confirmé)

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

4.2.4 FalArHeader

```
FalArHeader::= Unsigned8 {
  -- bit 8-7          ProtocolVersion
  -- bit 6-4          Protocol Identifier          Révision de la syntaxe abstraite des identificateurs et règles de
codage
   --bit 3-1          PDU Identifier               Identifie un type de PDU dans un identificateur de protocole
```

4.2.5 InvokeID

```
InvokeID::= Unsigned8                              Identifie cette invocation du service.
```

4.2.6 ServiceType

```
ServiceType::= Unsigned16                          Contient les balises spécifiques au contexte pour le corps de PDU
```

4.3 Syntaxe abstraite du corps de PDU

4.3.1 PDU "ConfirmedServiceRequest"

```
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,
      WriteAndRead-Request            [4]    IMPLICIT      WriteAndRead-RequestPDU,
      WriteAndReadMultiple-Request    [5]    IMPLICIT      WriteAndReadMultiple-RequestPDU
}
```

4.3.2 PDU "ConfirmedServiceResponse"

```
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,
      WriteAndRead-Response           [4]    IMPLICIT      WriteAndRead-ResponsePDU,
      WriteAndReadMultiple-Response   [5]    IMPLICIT      WriteAndReadMultiple-ResponsePDU
}
```

4.3.3 PDU "UnconfirmedServiceRequest"

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

}
```

4.3.4 Informations d'erreur

4.3.4.1 Type d'erreur

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

4.3.4.2 Classe d'erreurs

```
    ErrorClass::= CHOICE {
        noError                       [0]    IMPLICIT      Integer16 {
                                               normale                          (0),
                                               Autres                           *1)
        }
        Définition                    [1]    IMPLICIT      Integer16 {
                                               Autre                            (0),
                                               object-undefined (objet non défini)  (1),
        }
        Ressource                     [2]    IMPLICIT      Integer16 {
                                               Autre                            (0),
                                               memory-unavailable (mémoire      *1)
                                               indisponible)
        }
        Service                       [3]    IMPLICIT      Integer16 {
```

```
                                    Autre                                   (0),
                                    pdu-size (taille PDU)                   (1),
                                    illegal-parameter (paramètre interdit)  *2)
        }
        Accès                [4]    IMPLICIT    Integer16 {
                                    Autre                                   (0),
                                    object-access-denied (accès à l'objet   (1),
                                    refusé)
                                    invalid-address (adresse invalide)      (2),
                                    object-access-unsupported (accès à      (3),
                                    l'objet non pris en charge)
                                    object-non-existent (objet inexistant)  (4),
        }
        Autres               [5]    IMPLICIT    Integer16 {
                                    Autre                                   *0)
        }
}
```

4.3.4.3 Code d'état

Les codes de statut de la primitive "confirmed response" (réponse confirmée) sont énumérés dans le Tableau 2.

Tableau 2 – Code d'état de la primitive "confirmed response" (réponse confirmée)

Code d'erreur	Type d'erreur	Détails et causes de l'erreur
0x00.	No Error (aucune erreur)	Aucune erreur
0x01.	Service type Error (erreur de type service)	Type de service inconnu
0x02.	Object Identifier Error (erreur d'identificateur d'objet)	Accès à l'objet non pris en charge
0x03.	Data type error (erreur de type de données)	Type de données autre que celui pris en charge
0x04.	Object Offset Error (erreur de décalage d'objet)	La demande dépasse la zone admise pour chaque objet
0x05.	Data Length error (erreur de longueur de données)	La demande dépasse la plage maximale de règles Ethernet à lire ou écrire à la fois

4.4 Unités de données de protocole (PDU, Protocol Data Units) relatives aux éléments de service application (ASE)

4.4.1 PDU de l'ASE du processus d'application

4.4.1.1 PDU "Identify-Request"

Identify-Request PDU::= NULL

4.4.1.2 PDU "Identify-Response"

Identify-ResponsePDU::= SEQUENCE {
Service status
Code d'état
Identifiant d'entité DL
Adresse MAC
Informations de port
Version de Protocole
Reserved8.
Type de l'appareil
Description de dispositif
Hardware-Version (Version de l'équipement matériel)
Numéro de série
Software-Version (Version de logiciel)
Software-Date (Date du logiciel)
ID du fournisseur
Code du produit
}

4.4.1.3 PDU "Status-Request"

Status-Request PDU::= NULL

4.4.1.4 PDU "Status-Response"

Status-ResponsePDU::= SEQUENCE {
Service status
Code d'état
Indicateurs de dispositif
Etat du dispositif
tx_cnt_normal (Compte des normaux en émission)
tx_cnt_all (Compte du total en émission)
rx_cnt_normal (Compte des normaux en réception)
rx_cnt_all (Compte du total en réception)
relay_cnt_normal (Compte des normaux relayés)
relay_cnt_all (Compte du total relayé)
}

4.4.1.5 Service status

Service status::= Unsigned16 — donne des informations relatives au résultat d'exécution du service.

4.4.1.6 Code d'état

Status code::= Unsigned16 — Voir 4.3.4.3.

4.4.1.7 Identifiant d'entité DL

dl-address::= Unsigned16 — Voir l'IEC 61158-4-21, 4.6.5.2.

4.4.1.8 Adresse MAC

MAC address::= Unsigned48 — Voir l'IEC 61158-4-21, 4.6.5.8.

4.4.1.9 Informations de port

Port information::= Unsigned16 — Voir l'IEC 61158-4-21, 4.6.5.9.

4.4.1.10 Version de Protocole

Protocol version::= Unsigned8 — Voir l'IEC 61158-4-21, 4.6.5.10.

4.4.1.11 Reserved8.

Reserved8::= Unsigned8 — Reservé

4.4.1.12 Type de l'appareil

Device type::= Unsigned16 — Voir l'IEC 61158-4-21, 4.6.5.11.

4.4.1.13 Description de dispositif

Device description::= VISIBLE_STRING[16] — Voir l'IEC 61158-4-21, 4.6.5.12.

4.4.1.14 Hardware-Version (Version de l'équipement matériel)

Hardware-Version::= Unsigned16 — Contient la version matérielle de l'appareil.

4.4.1.15 Numéro de série

Serial Number::= Unsigned16 — Contient le numéro de série de l'appareil.

4.4.1.16 Software-Version (Version de logiciel)

Software-Version::= Unsigned16 — Contient la version logicielle de l'appareil.

4.4.1.17 Software-Date (Date du logiciel)

Software-Date::= Unsigned16 — Contient la date du logiciel de l'appareil.

4.4.1.18 ID du fournisseur

Vendor ID::= Unsigned16 — Contient l'identificateur du fournisseur de l'appareil.

4.4.1.19 Code du produit

Product Code::= Unsigned16 — Contient le code de produit de l'appareil.

4.4.1.20 Indicateurs de dispositif

Device flags::= Unsigned16 — Voir l' IEC 61158-4-21, 4.6.5.3.

4.4.1.21 Etat du dispositif

Device state::= Unsigned16 — Voir l' IEC 61158-4-21, 4.6.5.4.

4.4.1.22 tx_cnt_normal (Compte des normaux en émission)

tx_cnt_normal ::= Unsigned32 — La valeur du compte en émission des paquets normaux.

4.4.1.23 tx_cnt_all (Compte du total en émission)

tx_cnt_all::= Unsigned32 — La valeur du compte en émission des paquets erronés et des paquets normaux.

4.4.1.24 rx_cnt_normal (Compte des normaux en réception)

rx_cnt_normal::= Unsigned32 — La valeur du compte en réception des paquets normaux.

4.4.1.25 rx_cnt_all (Compte du total en réception)

rx_cnt_all::= Unsigned32 — La valeur du compte en réception des paquets erronés et des paquets normaux.

4.4.1.26 relay_cnt_normal (Compte des normaux relayés)

relay_cnt_normal ::= Unsigned32 — La valeur du compte relayé de paquets normaux

4.4.1.27 relay_cnt_all (Compte du total relayé)

relay_cnt_all::= Unsigned32 — La valeur du compte relayé des paquets erronés et des paquets normaux.

4.4.2 PDU relatives à l'ASE d'objet de données de service

4.4.2.1 Les PDU de service Read

Read-RequestPDU::= SEQUENCE {
  Object List Count (Compte de liste d'objets)
  Reserved16.
  Identificateur d'objet (k)
  Type de données
  offset
  pile double
  Identificateur d'objet (m)
  Type de données
  offset
  pile double
  … — (demandes de lecture ultérieures)
}

Read-ResponsePDU::= SEQUENCE {
  Service status
  Code d'état
  Object List Count (Compte de liste d'objets)
  Identificateur d'objet (k)
  Type de données
  offset
  pile double
  données
  Identificateur d'objet (m)
  Type de données
  offset
  pile double
  données
  … — (réponses de lecture ultérieures)
}

4.4.2.2 Les PDU de service Write

Write-RequestPDU::= SEQUENCE {
  Object List Count (Compte de liste d'objets)
  Reserved16.
  Identificateur d'objet (k)
  Type de données
  offset
  pile double
  données
  Identificateur d'objet (m)
  Type de données
  offset
  pile double
  données
  (demandes d'écriture ultérieure)
}

Write-ResponsePDU::= SEQUENCE {
  Service status
  Code d'état
}

4.4.2.3 Les PDU de service WriteAndRead

```
WriteAndRead-RequestPDU::= SEQUENCE {
   Compte de liste d'objets       (Read (lecture))
   Reserved16.
   Identificateur d'objet (k)
   Type de données
   offset
   pile double
   Identificateur d'objet (m)
   Type de données
   offset
   pile double
   ...                                    — (demandes de lecture ultérieures)
   Compte de liste d'objets       (Write (écriture))
   Reserved16.
   Identificateur d'objet (k)
   Type de données
   offset
   pile double
   données
   Identificateur d'objet (m)
   Type de données
   offset
   pile double
   données
   ...(demandes d'écriture ultérieure)
}
WriteAndRead-ResponsePDU::= SEQUENCE {
   État du service(Write (écriture))
   Code d'état
   État du service(Read (lecture))
   Code d'état
   Object List Count (Compte de liste d'objets)
   Reserved16.
   Identificateur d'objet (k)
   Type de données
   offset
   pile double
   données
   Identificateur d'objet (m)
   Type de données
   offset
   pile double
   données
   (demandes de lecture ultérieures)
}
```

4.4.2.4 PDU de service WriteAndReadMultiple

WriteAndReadMultiple-RequestPDU::= SEQUENCE {
Compte de liste d'appareils
Reserved16.
Device Address
Device Offset (décalage d'appareil)
Compte de liste d'objets (Read (lecture))
Reserved16.
Identificateur d'objet (k)
Type de données
offset
pile double
Identificateur d'objet (m)
Type de données
offset
pile double
(demandes de lecture ultérieures)
Compte de liste d'objets (Write (écriture))
Reserved16.
Identificateur d'objet (k)
Type de données
offset
pile double
données
Identificateur d'objet (m)
Type de données
offset
pile double
données
(demandes d'écriture ultérieure)
}

WriteAndReadMultiple-ResponsePDU::= SEQUENCE {
Compte de liste d'appareils
Reserved16.
Device Address
Device Offset (décalage d'appareil)
État du service(Write (écriture))
Code d'état
État du service(Read (lecture))
Code d'état
Object List Count (Compte de liste d'objets)
Reserved16.
Identificateur d'objet (k)
Type de données
offset
pile double
données
Identificateur d'objet (m)
Type de données
offset
pile double
données
(demandes de lecture ultérieures)

4.4.2.5 Service status

Service status::= Unsigned16 — donne des informations relatives au résultat d'exécution du service.

4.4.2.6 Code d'état

Status code::= Unsigned16 — Voir 4.3.4.3.

4.4.2.7 Object List Count (Compte de liste d'objets)

Object list count::= Unsigned16 — Nombre d'objets.

4.4.2.8 Reserved16.

Reserved16::= Unsigned16 — Réservé.

4.4.2.9 Identificateur d'objet

Object identifier::= Unsigned16 — Spécifie une entrée de l'objet d'appareil.

4.4.2.10 Type de données

Data type::= Unsigned16 — Contient l'identificateur numérique du type de données.

4.4.2.11 Décalage

offset ::= Unsigned32 — Fournit le décalage relatif au début de l'objet.

4.4.2.12 Longueur

length::= Unsigned32 — Le nombre d'octets du service qui doivent être lus ou écrits.

4.4.2.13 données

data::= Any — type et longueur dépendant de l'application;
ce champ doit assurer un alignement unsigned32;
la longueur totale de la trame doit être conforme aux règles Ethernet.

4.4.3 PDU relatives à l'ASE d'objet de données de processus

4.4.3.1 PDU de service TB-transfer

```
TB-transfer PDU::= SEQUENCE {
      TBArep,      -- Numéro de bloc
      TBData       -- Contenu du segment basé sur un temporisateur (TB, time-based)
}
```

4.4.3.2 TBArep

TBArep::= Unsigned16 — Numéro de bloc

4.4.3.3 TBData

```
TBData::= SEQUENCE {
   blen     Unsigned16,          — Longueur en octets TB
   payload-data::=Any            — Contenu TB
```

4.4.3.4 PDU de service COS-transfer

```
COS-transfer PDU::= SEQUENCE {
      COSArep,    -- Numéro de bloc
      COSData     -- contenu du segment de changement d'état (COS, Change-Of-State)
}
```

4.4.3.5 COSArep

COSArep::= Unsigned16 — Numéro de bloc

4.4.3.6 COSData

```
COSData::= SEQUENCE {
   blen     Unsigned16,          — Longueur en octets du COS
   payload-data::=Any            — Contenu du COS
```


5 Transfer Syntax (Syntaxe de transfert)

5.1 Vue d'ensemble du codage

Les PDU de FAL codées doivent avoir un format uniforme. Elles doivent être constituées de deux parties principales: l'en-tête de l'APDU et le corps de l'APDU représentés sur la Figure 2.

*1)	*1)	*2)	(n) --- octets
Champ FalArHeader	(InvokeID)	ServiceType	Paramètres spécifiques au service
<---------- En-tête de l'APDU ---------->			<---------- Corps de l'APDU ---------->

NOTE La présence du champ InvokeID dépend du type d'APDU.

Figure 2 – Vue d'ensemble des APDU

Pour réaliser une APDU efficace tout en gardant un codage flexible, différentes règles de codage sont utilisées pour l'en-tête de l'APDU et le corps de l'APDU.

NOTE Le service DLL fournit un paramètre DLSDU qui indique la longueur de l'APDU. Par conséquent, les informations de longueur d'APDU ne sont pas incluses dans l'APDU.

5.2 Encodage de l'en-tête des unités APDU

L'en-tête de l'APDU est toujours présente dans toutes les APDU conformes au présent document. Elle est constituée de trois champs: le champ FalArHeader, le champ InvokeID et le champ ServiceType comme représenté à la Figure 2.

5.2.1 Codage du champ FalArHeader

Toutes les PDU de la FAL ont l'en-tête de PDU commun appelé FalArHeader. Le FalArHeader identifie la syntaxe abstraite, la syntaxe de transfert et chacune des PDU. Le Tableau 3 définit la manière dont cet en-tête doit être utilisé.

Tableau 3 – Codage d'un champ FalArHeader

Position de bit de FalArHeader			Type de PDU	Version de Protocole
8 7	6 5 4	3 2 1		
01	001	000	ConfirmedSend-CommandPDU	Version 1
01	001	100	ConfirmedSend-ResponsePDU	Version 1
01	010	000	UnconfirmedSend-CommandPDU	Version 1
NOTE Tous les autres points de code sont réservés à des protocoles supplémentaires et des révisions futures.				

5.2.2 Codage du champ InvokeID

Le champ InvokeID doit être présent s'il est indiqué dans la syntaxe abstraite. Sinon, ce champ ne doit pas être présent. S'il est présent, le paramètre InvokeID fourni par une primitive de service doit être placé dans ce champ.

5.2.3 Codage du champ Type

Le type de service d'une APDU est codé dans le champ Type qui est toujours le troisième octet de l'APDU.

Tous les bits du champ Type sont utilisés pour coder le type de service, comme suit:

a) les types de service doivent être codés dans les bits 16 à 1 du champ Type, le bit 16 étant le MSB et le bit 1, le bit de poids faible (LSB, Least Significant Bit). La plage du type de service doit être comprise entre 0 et 65 534;

b) la valeur de 65 535 est réservée aux extensions futures au présent document;

c) le type de service est spécifié dans la syntaxe abstraite comme une valeur entière positive.

La Figure 3 illustre le codage du champ Type.

Figure 3 – Champ Type

5.3 Encodage du corps des unités APDU

5.3.1 Généralités

Les règles de codage de la FAL sont basées sur les termes et conventions définis dans l'ISO/IEC 8825--1. Le codage se compose de trois composants dans l'ordre suivant:

a) octet d'identificateur;
b) octet(s) de longueur;
c) octet(s) de contenu.

NOTE Les octets d'identification et les octets de longueur de contenu n'existent pas dans le Type 21.

5.4 Codage des types de données

5.4.1 Description générale des types de données et des règles de codage

Le format et la signification de ces données doivent être connus du producteur et du ou des consommateurs pour qu'ils puissent échanger des données qui aient du sens. À cette fin, le présent modèle de document utilise le concept des types de données.

Les règles de codage définissent la représentation des valeurs des types de données et la syntaxe de transfert pour les représentations. Les valeurs sont représentées sous la forme de séquences binaires. Les séquences de bits sont transférées en séquences d'octets.

5.4.2 Syntaxe de transfert des séquences binaires

Une suite de bits est réordonnée en une séquence d'octets pour la transmission. La notation hexadécimale est utilisée pour les octets (voir l'ISO/IEC 9899). Soit b = $b_0 ... b_{n-1}$ une séquence binaire et k un entier non négatif tel que 8(k-1) < n ≤ 8k. Alors, b est transféré dans k octets assemblés comme indiqué dans le Tableau 4. Les bits bi, i > n de l'octet le plus élevé sont des bits sans importance.

Tableau 4 – Syntaxe de transfert des séquences binaires

Numéro d'octet	1.	2.	k)
	b_0 .. b_7	b_8 .. b_{15}	b_{8k-8} .. b_{8k-1}

L'octet 1 est transmis le premier, et l'octet k le dernier. La suite de bits est transférée comme suit sur tout le réseau (l'ordre d'émission au sein d'un octet est déterminé par l'ISO/IEC 8802-3):

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

EXEMPLE:

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

La séquence de bits b = b_0 .. b_9 = 0011 1000 01_b représente un entier de type Unsigned10, de valeur 21Ch, et est transférée sous la forme de deux octets: d'abord 1Ch, puis 02h.

5.4.3 Codage d'une valeur Boolean

Les données fondamentales de type BOOLÉEN peuvent prendre des valeurs "VRAI" ou "FAUX".

Les valeurs sont représentées comme des séquences binaires de longueur 1. La valeur TRUE (vrai) est représentée par un 1 et la valeur FALSE (FAUX) est représentée par un 0.

Une valeur booléenne BOOLEAN doit être transférée sur le réseau comme UNSIGNED8 de valeur 1 (TRUE) ou 0 (FALSE). Les séquences de booléens BOOLEAN peuvent être empaquetées en un UNSIGNED8. Les séquences d'éléments de type BOOLEAN et BIT peuvent également être condensées en un UNSIGNED8.

5.4.4 Encodage d'une valeur entière Unsigned (non signée)

Les données d'un type de données de base Unsignedn contiennent des valeurs dans les entiers non négatifs. La plage de valeurs est 0, ..., 2^n-1. Les données sont représentées sous forme de séquences de bits de longueur n.

La séquence binaire

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

comporte la valeur

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

Il est à noter que la séquence de bit commence à gauche par l'octet de poids faible.

EXEMPLE: La valeur $266_d = 10A_h$ avec le type de données UNSIGNED16 est transférée dans deux octets à travers le bus, d'abord $0A_h$, puis 01_h.

Les types de données UNSIGNEDn sont spécifiés dans le Tableau 5.

Tableau 5 – Syntaxe de transfert du type de données Unsignedn

Numéro d'octet	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 Encodage d'une valeur entière Signed (signée)

Les données d'un type de données de base INTEGERn contiennent des valeurs entières. La plage de valeurs s'étend de -2^{n-1} à $2^{n-1}-1$. Les données sont représentées sous la forme de suites de bits de longueur n. La suite de bits

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

comporte la valeur

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

et, après deux calculs arithmétiques,

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

Il est à noter que la séquence de bit commence à gauche par le bit de poids faible.

EXEMPLE La valeur –266 = 0xFEF6 de type Integer16 est transférée sous la forme de deux octets: d'abord 0xF6, ensuite 0xFE.

Le transfert des types de données INTEGERn est tel que spécifié dans le Tableau 6.

Tableau 6 – Syntaxe de transfert du type de données INTEGERn

Numéro d'octet	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 Codage d'une valeur en virgule Flottante

Les données des types de données de base REAL32 et REAL64 ont des valeurs en nombres réels.

Le type de données REAL32 est représenté sous forme de séquences 32 bits. Le codage des valeurs suit la norme IEEE 754-2008 pour une virgule flottante en simple précision.

Le type de données REAL64 est représenté sous forme de séquences 64 bits. Le codage des valeurs suit la norme IEEE 754-2008 pour les nombres à virgule flottante en double précision.

Une séquence 32 bits a une valeur (nombre réel indépendant de zéro, ± 0, ± _) ou n'est pas un nombre (NaN).

La séquence binaire

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

se voit attribuer la valeur (nombre fini non nul)

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

où

S = b_{31} est le signe.

E = $b_{23} \times 2^0 + \cdot + b_{30} \times 2^7$, 0 < E < 255, est l'exposant sans biais.

F = $2^{-23} \times (b_0 \times 2^0 + b_1 \times 2^1 + \approx + b_{22} \times 2^{22})$ est la partie fractionnaire du nombre.

E = 0 est utilisé pour représenter ± 0. E = 255 est utilisé pour représenter les valeurs infinies et les NaN.

La séquence binaire doit commencer à gauche par l'octet de poids faible.

5.4.7 Codage d'une valeur de chaîne d'octets

Le type de données OCTET_STRINGlength est défini comme suit, où "length" est la longueur de la chaîne d'octets.

ARRAY [length] OF UNSIGNED8 OCTET_STRINGlength

5.4.8 Codage d'une valeur de chaîne visible

VISIBLE_CHAR sont 0_h et la plage est comprise entre 20_h et $7E_h$. Les données sont interprétées
en caractères codés 7 bits conformément à l'ISO/IEC 646 et "length" est la longueur de la chaîne visible.

UNSIGNED8 VISIBLE_CHAR

ARRAY [length] OF VISIBLE_CHAR VISIBLE_STRINGlength

Aucun 0_h n'est nécessaire pour terminer la chaîne.

5.4.9 Encodage d'une chaîne de valeur Unicode

Le type de données UNICODE_STRINGlength est défini comme suit, où "length" est la longueur de la chaîne unicode.

ARRAY [length] OF UNSIGNED16 UNICODE_STRINGlength

5.4.10 Codage d'une heure de la valeur du jour

Le type de données TimeOfDay représente le temps absolu. TimeOfDay est représenté comme une séquence de 48 bits de longueur.

Le composant "ms" est le temps en millisecondes après minuit. Le composant "days" est le nombre de jours à partir du 01-01-1984.

```
STRUCT of
   UNSIGNED28      ms,
   VOID4           réservé,
   UNSIGNED16      jours
TIME_OF_DAY
```

Le codage est comme représenté sur la 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)	nombre de ms depuis minuit
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)	nombre de jours à partir du 01-01-1984
6	2 (7)	2 (6)	2 (5)	2 (4)	2.3	2 (2)	2 (1)	2 (0)	
	bit de poids fort								

Figure 4 – Encodage d'une valeur Time of Day

5.4.11 Codage d'une valeur d'écart de temps

Le type de données TimeDifference représente un écart de temps ou une durée. TimeDifference est représenté comme une séquence de 48 bits de longueur.

Les écarts de temps sont des sommes de nombres de jours et de millisecondes. Le composant "ms" est le nombre de millisecondes. Le composant "days" est le nombre de jours.

```
STRUCT of
   UNSIGNED28      ms,
   VOID4           réservé,
   UNSIGNED16      jours
TIME_DIFFERENCE
```

Le codage est comme représenté sur la 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)	jours
6	2 (7)	2 (6)	2 (5)	2 (4)	2.3	2 (2)	2 (1)	2 (0)	
	bit de poids fort								

Figure 5 – Encodage d'une valeur Time Difference

6 Diagrammes d'états de protocole FAL

Les interfaces avec les services FAL et les machines de protocole sont spécifiées dans le présent document.

NOTE Les diagrammes d'états spécifiés dans le présent document et les machines de protocole de relation entre applications définies ci-dessous définissent uniquement les événements valides pour chacun. Le traitement d'événements invalides est une responsabilité locale.

Le comportement de la FAL est décrit par les machines de protocole représentées sur la Figure 6. Les trois types de machines de protocole sont: les machines de protocole de service FAL (FSPM, FAL Service Protocol Machines), les machines de protocole de relation entre applications (ARPM, Application Relationship Protocol Machines) et les machines de protocole de mapping DLL (DMPM, DLL Mapping Protocol Machines). Des ensembles spécifiques de ces machines de protocole sont définis pour différents types de points d'extrémité de relation entre applications (AREP). La Figure 6 montre également les primitives échangées entre les machines de protocole.

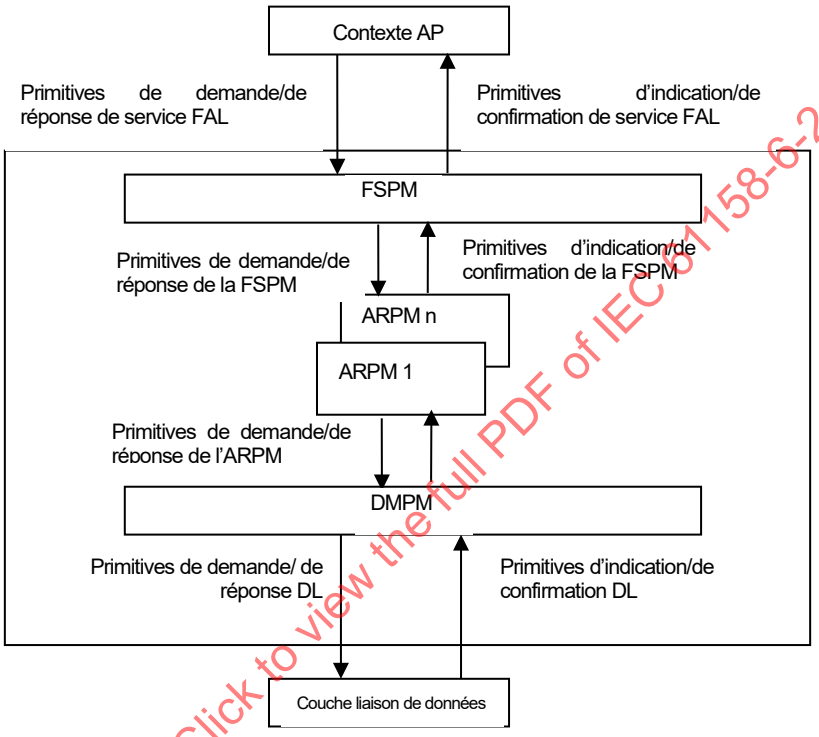


Figure 6 – Primitives échangées entre les machines de protocoles

La FSPM est responsable de ce qui suit:

a) Accepter les primitives de service émises par l'utilisateur de service FAL et les convertir en primitives internes FAL.

b) sélectionner un diagramme d'états ARPM approprié en fonction des paramètres d'identifiant AREP fournis par le contexte AP et envoyer les primitives FAL internes à la machine ARPM choisie;

c) accepter les primitives internes FAL de l'ARPM et les convertir en primitives de service pour l'AP-Context;

d) fournir les primitives de service FAL à l'AP-Context d'après le paramètre Identificateur d'AREP associé aux primitives.

L'ARPM est responsable de ce qui suit:

a) accepter les primitives internes FAL de la FSPM et créer et envoyer d'autres primitives internes FAL à la FSPM ou à la DMPM, en fonction de l'AREP et des types de primitive;

b) accepter les primitives internes FAL de la DMPM et les envoyer à la FSPM sous une forme convertie pour la FLSPM;

c) établir ou libérer l'AR spécifiée si les primitives concernent respectivement les services Establish ou Abort.