

Industrial communication networks — Fieldbus specifications —

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

ICS 25.040.40; 35.100.70

National foreword

This British Standard is the UK implementation of EN 61158-6-9:2008. It is identical with IEC 61158-6-9:2007. Together with all of the other sections of BS EN 61158-6, it supersedes BS EN 61158-6:2004 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee AMT/7, Industrial communications — Process measurement and control, including Fieldbus.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2008

© BSI 2008

ISBN 978 0 580 61629 7

Amendments/corrigenda issued since publication

Date	Comments

English version

**Industrial communication networks -
Fieldbus specifications -
Part 6-9: Application layer protocol specification -
Type 9 elements
(IEC 61158-6-9:2007)**

Réseaux de communication industriels -
Spécifications des bus de terrain -
Partie 6-9: Spécification des services
des couches d'application -
Éléments de type 9
(CEI 61158-6-9:2007)

Industrielle Kommunikationsnetze -
Feldbusse -
Teil 6-9: Protokollspezifikation
des Application Layer
(Anwendungsschicht) -
Typ 9-Elemente
(IEC 61158-6-9:2007)

This European Standard was approved by CENELEC on 2008-02-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Central Secretariat: rue de Stassart 35, B - 1050 Brussels

Foreword

The text of document 65C/476/FDIS, future edition 1 of IEC 61158-6-9, prepared by SC 65C, Industrial networks, of IEC TC 65, Industrial-process measurement, control and automation, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 61158-6-9 on 2008-02-01.

This and the other parts of the EN 61158-6 series supersede EN 61158-6:2004.

With respect to EN 61158-6:2004 the following changes were made:

- deletion of Type 6 fieldbus for lack of market relevance;
- addition of new fieldbus types;
- partition into multiple parts numbered 6-2, 6-3, ...6-20.

The following dates were fixed:

- latest date by which the EN has to be implemented
at national level by publication of an identical
national standard or by endorsement (dop) 2008-11-01
- latest date by which the national standards conflicting
with the EN have to be withdrawn (dow) 2011-02-01

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

Annex ZA has been added by CENELEC.

Endorsement notice

The text of the International Standard IEC 61158-6-9:2007 was approved by CENELEC as a European Standard without any modification.

CONTENTS

INTRODUCTION	6
1 Scope	7
1.1 General	7
1.2 Specifications	7
1.3 Conformance	7
2 Normative references	8
3 Terms, definitions, symbols, abbreviations and conventions	8
3.1 Terms and definitions from other ISO/IEC standards	8
3.2 IEC/TR 61158-1 terms	9
3.3 Abbreviations and symbols	13
3.4 Conventions	14
3.5 Conventions used in state machines	14
4 Abstract syntax	15
4.1 FAL-AR PDU abstract syntax	15
4.2 Abstract syntax of PDUBody	18
4.3 Type definitions for ASEs	21
4.4 Abstract syntax of data types	26
5 Transfer syntax	27
5.1.1 General	27
5.1.2 Coding rules	27
5.1.3 Structure of the identification information	28
6 Structure of FAL protocol state machines	37
7 AP-Context state machines	39
7.1 VCR PM structure	39
7.2 VCR PM state machine	39
8 FAL service protocol machine (FSPM)	52
8.1 General	52
8.2 FSPM state tables	52
8.3 Functions used by FSPM	55
8.4 Parameters of FSPM/ARPM primitives	55
9 Application relationship protocol machines (ARPMs)	55
9.1 AREP mapping to data-link layer	55
9.2 Application relationship protocol machines (ARPMs)	65
9.3 AREP state machine primitive definitions	81
9.4 AREP state machine functions	84
10 DLL mapping protocol machine (DMPM)	85
10.1 DMPM States	85
10.2 DMPM state table	85
10.3 Primitives exchanged between data-link layer and DMPM	93
10.4 Functions used by DMPM	96
Annex ZA (normative) Normative references to international publications with their corresponding European publications	99
Bibliography	98
Figure 1 – Insertion of identification information in the FMS PDU	27
Figure 2 – Identification	

Figure 3 – Coding with identification	29
Figure 4 – Coding without identification.....	29
Figure 5 – Representation of the value true	29
Figure 6 – Representation of the value false	30
Figure 7 – Coding of data of data type Integer16.....	30
Figure 8 – Coding of data of data type Unsigned16	31
Figure 9 – Coding of data of data type Floating Point.....	31
Figure 10 – Coding of data of data type Visible String.....	32
Figure 11 – Coding of data of data type Octet String	32
Figure 12 – Coding of data of type Date	33
Figure 13 – Coding of data of data type Time-of-day	34
Figure 14 – Coding of data of data type Time-difference	34
Figure 15 – Coding of data of data type Bit String	35
Figure 16 – Coding of data of data type Time-value	35
Figure 17 – Coding of data of user data definitions with identifier	36
Figure 18 – Coding of data of user data definitions without identifier	36
Figure 19 – Coding of ID info for a SEQUENCE	36
Figure 20 – Relationships among protocol machines and adjacent layers	38
Figure 21 – Relationships among protocol machines and adjacent layers	39
Figure 22 – VCR state machine	40
Figure 23 – State transition diagram of FSPM.....	52
Figure 24 – State transition diagram of the QUU ARPM	66
Figure 25 – State transition diagram of QUB ARPM	68
Figure 26 – State transition diagram of the BNU ARPM.....	77
Figure 27 – State transition diagram of DMPM	85
Table 1 – Conventions used for state machines	14
Table 2 – Coding for Date type	33
Table 3 – AP-VCR state machine transactions	41
Table 4 – Primitives issued by FAL-User to VCR PM.....	50
Table 5 – Primitives issued by VCR PM to FAL-User.....	50
Table 6 – Primitives issued by VCR PM to FSPM	51
Table 7 – Primitives issued by FSPM to VCR PM	52
Table 8 – FSPM state table – sender transactions	53
Table 9 – FSPM state table – receiver transactions.....	54
Table 10 – Function SelectArep().....	55
Table 11 – Parameters used with primitives exchanged between FSPM and ARPM	55
Table 12 – QUU ARPM states	65
Table 13 – QUU ARPM state table – sender transactions.....	66
Table 14 – QUU ARPM state table – receiver transactions.....	67
Table 15 – QUB ARPM states	68
Table 16 – QUB ARPM state table – sender transactions.....	69
Table 17 – QUB ARPM state table – receiver transactions.....	70

Table 18 – BNU ARPM states	76
Table 19 – BNU ARPM state table – sender transactions	77
Table 20 – BNU ARPM state table – receiver transactions	79
Table 21 – Primitives issued from ARPM to DMPM	82
Table 22 – Primitives issued by DMPM to ARPM.....	82
Table 23 – Parameters used with primitives exchanged between ARPM and DMPM	83
Table 24 – Function GetArepld()	84
Table 25 – Function BuildFAS-PDU	84
Table 26 – Function FAS_Pdu_Type	84
Table 27 – Function AbortIdentifier	84
Table 28 – Function AbortReason	84
Table 29 – Function AbortDetail	85
Table 30 – DMPM state descriptions	85
Table 31 – DMPM state table – sender transactions.....	86
Table 32 – DMPM state table – receiver transactions.....	88
Table 33 – Primitives exchanged between data-link layer and DMPM	94
Table 34 – Function PickArep	96
Table 35 – Function FindAREP	97
Table 36 – Function LocateQubArep	97
Table 37 – Function SetIdentifier()	97

INTRODUCTION

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

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

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

1 Scope

1.1 General

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

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to type 9 fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

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

- a) the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,
- b) the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,
- c) the application context state machine defining the application service behavior visible between communicating application entities; and
- d) the application relationship state machines defining the communication behavior visible between communicating application entities; and.

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

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

This standard specifies the protocol of the Type 9 IEC fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI Application Layer Structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-9.

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

1.3 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

Conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

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

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

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

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

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

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

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

ISO/IEC 8824, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*

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

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

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

3 Terms, definitions, symbols, abbreviations and conventions

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

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 Terms and definitions from ISO/IEC 7498-1

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

- h) application transaction
- i) presentation context
- j) real open system
- k) transfer syntax

3.1.2 Terms and definitions from ISO/IEC 9545

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

3.1.3 Terms and definitions from ISO/IEC 8824

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

3.1.4 Terms and definitions from ISO/IEC 8825

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

3.2 IEC/TR 61158-1 terms

The following IEC/TR 61158-1 terms apply.

3.2.1 application

function or data structure for which data is consumed or produced

3.2.2 application layer interoperability

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

3.2.3 application object

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

NOTE: Multiple types of application object classes may be defined

3.2.4 application process

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

3.2.5 application process identifier

distinguishes multiple application processes used in a device

3.2.6 application process object

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

3.2.7 application process object class

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

3.2.8 application relationship

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

3.2.9 application relationship application service element

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

3.2.10 application relationship endpoint

context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship. Each application process involved in the application relationship maintains its own application relationship endpoint

3.2.11 attribute

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

3.2.12 behaviour

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

3.2.13 class

a set of objects, all of which represent the same kind of system component. A class is a generalisation of the object; a template for defining variables and methods. All objects in a class are identical in form and behaviour, but usually contain different data in their attributes

3.2.14 class attributes

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

3.2.15 class code

a unique identifier assigned to each object class

3.2.16 class specific service

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

3.2.17 client

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

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

3.2.18 conveyance path

unidirectional flow of APDUs across an application relationship

3.2.19 cyclic

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

3.2.20 dedicated AR

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

3.2.21 device

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

3.2.22 device profile

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

3.2.23 dynamic AR

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

3.2.24 endpoint

one of the communicating entities involved in a connection

3.2.25 error

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

3.2.26 error class

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

3.2.27 error code

identification of a specific type of error within an error class

3.2.28 FAL subnet

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

3.2.29 logical device

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

3.2.30 management information

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

3.2.31 network

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

3.2.32 peer

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

3.2.33 pre-defined AR endpoint

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

3.2.34 pre-established AR endpoint

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

3.2.35 publisher

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

3.2.36 server

- a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request
- b) an object which provides services to another (client) object

3.2.37 service

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

3.2.38 subscriber

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

3.3 Abbreviations and symbols

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

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

3.4 Conventions

3.4.1 General concept

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

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

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

3.4.2 Conventions for class definitions

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

3.4.3 Abstract syntax conventions

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

3.5 Conventions used in state machines

The state machines are described in Table 1:

Table 1 – Conventions used for state machines

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

The conventions used in the state machines are as follows:

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

xxx A parameter name.

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

"xxx" Indicates fixed value.

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

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

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

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

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

&& Logical "AND"

|| Logical "OR"

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

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

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

Example:
If (condition)
actions
else
actions
endif

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

4 Abstract syntax

4.1 FAL-AR PDU abstract syntax

4.1.1 Top level definition

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

```

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

```

4.1.2 Confirmed send service

```

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

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

```

4.1.3 Unconfirmed send service

```

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

```

4.1.4 Unconfirmed acknowledge send service

```

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

```

4.1.5 InvokeID

InvokeID ::= Unsigned8

4.1.6 Establish service

MaxOSCC ::= Unsigned8

MaxOSCS ::= Unsigned8

MaxUCSC ::= Unsigned8

MaxUCSS ::= Unsigned8

CIU ::= Unsigned32

```

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

```

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

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

4.1.7 ConType

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

4.1.8 Data send acknowledge service

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

4.1.9 Protocol-code

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

4.1.10 Block-number

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

4.1.11 Block-length

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

4.1.12 Protocol-data

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

4.1.13 Address2 ARP

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

4.1.14 Destination-address

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

4.1.15 Source-address

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

4.1.16 Destination-node

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

4.1.17 Source-node

Source-Node ::= Unsigned8

4.1.18 Destination-subnode

Destination-Subnode ::= Unsigned8

4.1.19 Source-subnode

Source-Subnode ::= Unsigned8

4.2 Abstract syntax of PDUBody**4.2.1 Abort service**

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

4.2.2 ConfirmedServiceRequest

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

4.2.3 ConfirmedServiceResponse

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

4.2.4 ConfirmedServiceError

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

4.2.5 Error type

Error type as specified in 4.1.4.6.

4.2.6 Error Fi type

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

4.2.7 Error class

Error Class as specified in 4.1.4.7.

4.2.8 Unconfirmed PDUs

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

UnconfirmedSendPD-PDU ::= BIT STRING

4.2.9 Management ASE

4.2.9.1 Get attributes service

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

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

4.2.10 Application process ASE

4.2.10.1 Get status service

Status-RequestPDU ::= NULL

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

4.2.10.2 Identify service

Identify-RequestPDU ::= NULL

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

4.2.10.3 Initiate service

```

Initiate-RequestPDU ::= SEQUENCE {
  versionObjectDefinitionsCalling
  apDescriptorCalling
  accessProtectionSupportedCalling
  passwordAndAccessGroupsCalling
  configuredMaxPduSizeSending
  configuredMaxPduSizeReceiving
  listOfSupportedServicesCalling
}
Initiate-ResponsePDU ::= SEQUENCE {
  versionObjectDefinitionsCalled
  apDescriptorCalled
  accessPrivilegeSupportedCalled
  passwordAndAccessGroupsCalled
}
Initiate-ErrorPDU ::= SEQUENCE {
  errorCode
  other
  max-fal-pdu-size-insufficient
  service-not-supported
  version-obj-def-incompatible
  user-initiate-denied
  password-error
  profile-number-incompatible
},
  maxPduLengthSendingCalled
  maxPduLengthReceivingCalled
  listOfSupportedServicesCalled
}

```

[PRIVATE 0] IMPLICIT Integer16,
 [PRIVATE 1] IMPLICIT OctetString,
 [PRIVATE 2] IMPLICIT Ap_AccessProtectionSupported,
 [PRIVATE 3] IMPLICIT Ap_AccessControl,
 [PRIVATE 4] IMPLICIT Unsigned8,
 [PRIVATE 5] IMPLICIT Unsigned8,
 [PRIVATE 6] IMPLICIT Mn_PduSupportedMap

[PRIVATE 0] IMPLICIT Integer16,
 [PRIVATE 1] IMPLICIT OctetString,
 [PRIVATE 2] IMPLICIT Ap_AccessProtectionSupported,
 [PRIVATE 3] IMPLICIT Ap_AccessControl

[PRIVATE 0] IMPLICIT ENUMERATED {
 (0),
 (1),
 (2),
 (3),
 (4),
 (5),
 (6)
},
 [PRIVATE 1] IMPLICIT Unsigned8,
 [PRIVATE 2] IMPLICIT Unsigned8,
 [PRIVATE 3] IMPLICIT Mn_PduSupportedMap

4.2.10.4 Reject service

```

Reject-RequestPDU ::= SEQUENCE {
  original-invokeID
  reject-code
  pdu-size
}

```

[PRIVATE 0] IMPLICIT Integer8,
 [PRIVATE 1] IMPLICIT ENUMERATED {
 (5)

4.2.11 Function invocation ASE**4.2.11.1 Reset service**

```

Reset-RequestPDU ::= SEQUENCE {
  keyAttribute
}
Reset-ResponsePDU ::= NULL

```

Gn_KeyAttribute

4.2.11.2 Resume service

```

Resume-RequestPDU ::= SEQUENCE {
  keyAttribute
}
Resume-ResponsePDU ::= NULL

```

Gn_KeyAttribute

4.2.11.3 Start service

```

Start-RequestPDU ::= SEQUENCE {
  keyAttribute
}
Start-ResponsePDU ::= NULL

```

Gn_KeyAttribute

4.2.11.4 Stop service

```

Stop-RequestPDU ::= SEQUENCE {
  keyAttribute
}
Stop-ResponsePDU ::= NULL

```

Gn_KeyAttribute

4.2.12 Variable ASE

4.2.12.1 Information report service

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

4.2.12.2 Read service

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

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

4.2.12.3 Write service

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

Write-ResponsePDU ::= NULL
```

4.3 Type definitions for ASEs

4.3.1 AP ASE types

4.3.1.1 Ap_AccessProtectionSupported

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

4.3.1.2 Ap_AccessControl

```
Ap_AccessControl ::= BitString {
    password_Bit1      (8),
    password_Bit2      (7),
    password_Bit3      (6),
    password_Bit4      (5),
    password_Bit5      (4),
    password_Bit6      (3),
    password_Bit7      (2),
    password_Bit8      (1),
    access_Groups-1     (16),
    access_Groups-2     (15),
    access_Groups-3     (14),
    access_Groups-4     (13),
    access_Groups-5     (12),
    access_Groups-6     (11),
    access_Groups-7     (10),
    access_Groups-8     (9)
}
```

-- The Password (Unsigned8) is encoded as a bit string.

4.3.2 AR ASE types

4.3.2.1 Reason code and additional detail

4.3.2.1.1 Reason code

```
ReasonCode ::= Unsigned8
```

4.3.2.1.2 Additional detail

```
AdditionalDetail ::= OctetString
```

4.3.2.2 AREP

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

4.3.2.3 Abort type identifier

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

4.3.3 Function Invocation ASE types

4.3.3.1 Fi_AccessPrivilege

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

4.3.3.2 Fi_State

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

4.3.4 Management ASE types

4.3.4.1 Mn_PduSupportedMap

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

4.3.5 Variable ASE types

4.3.5.1 Vr_AccessPrivilege

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

-- The Password (Unsigned8) is encoded as a bit string.

4.3.6 General types

4.3.6.1 Gn_Deletable

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

4.3.6.2 Gn_KeyAttribute

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

4.3.6.3 Gn_Length

```
Gn_Length ::= Unsigned8
```

4.3.6.4 Gn_MoreFollows

```
Gn_MoreFollows ::= Boolean
```

4.3.6.5 Gn_NumericID

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

4.3.6.6 Gn_Name

```
Gn_Name ::= OctetString
```

4.3.6.7 Gn_ObjectClass

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

4.3.6.8 Gn_ObjectDefinition

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

4.3.6.9 Gn_Reusable

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

4.3.6.10 Gn_SubIndex

Gn_SubIndex ::= Unsigned8

4.3.6.11 Gn_TypeDescription

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

4.3.7 Object definitions**4.3.7.1 Top Level Definition**

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

4.3.7.2 ListHeader

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

4.3.7.3 DataTypeList

```

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

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

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

```

4.3.7.4 StaticList

```

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

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

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

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

```

4.3.7.5 FunctionInvocationDefinition

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

4.4 Abstract syntax of data types

4.4.1 Referenced data types

4.4.2 Notation for the Boolean type

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

4.4.3 Notation for the Integer types

```
Integer ::= INTEGER -- any integer
Integer8 ::= INTEGER (-128..+127) -- range  $-2^7 \leq i \leq 2^7-1$ 
Integer16 ::= INTEGER (-32768..+32767) -- range  $-2^{15} \leq i \leq 2^{15}-1$ 
Integer32 ::= INTEGER -- range  $-2^{31} \leq i \leq 2^{31}-1$ 
```

4.4.4 Notation for the Unsigned types

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

4.4.5 Notation for the Floating Point type

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

4.4.6 Notation for the BitString type

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

4.4.7 Notation for the OctetString type

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

4.4.8 Notation for VisibleString type

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

4.4.9 Notation for BinaryDate type

```
BinaryDate ::= OctetString7
```

4.4.10 Notation for TimeOfDay type

```
TimeOfDay ::= OctetString6
```

4.4.11 Notation for TimeDifference type

TimeDifference ::= OctetString6

4.4.12 Notation for TimeValue type

TimeValue ::= OctetString8

4.4.13 Notation for DL—Time-offset type

DL-Time-offsetType ::= OctetString

5 Transfer syntax

5.1.1 General

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

The conditions in the fieldbus area are the following:

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

5.1.2 Coding rules

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

Structure of a PDU is shown in Figure 1.

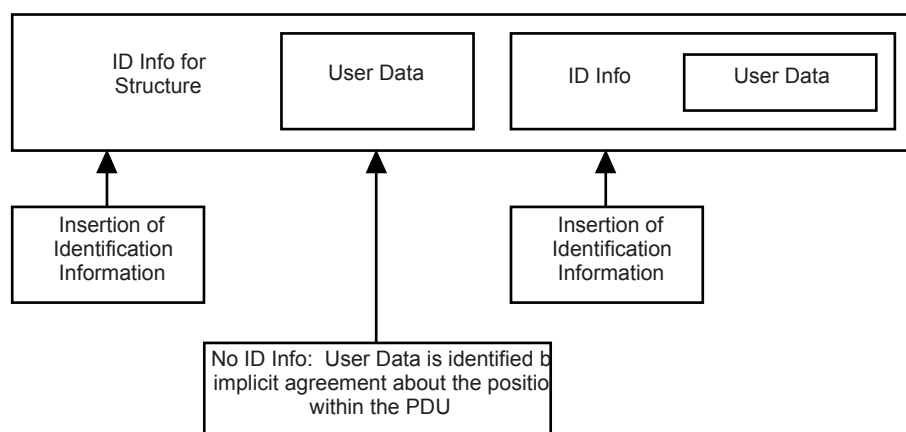


Figure 1 – Insertion of identification information in the FMS PDU

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

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

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

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

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

5.1.3 Structure of the identification information

5.1.3.1 General

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

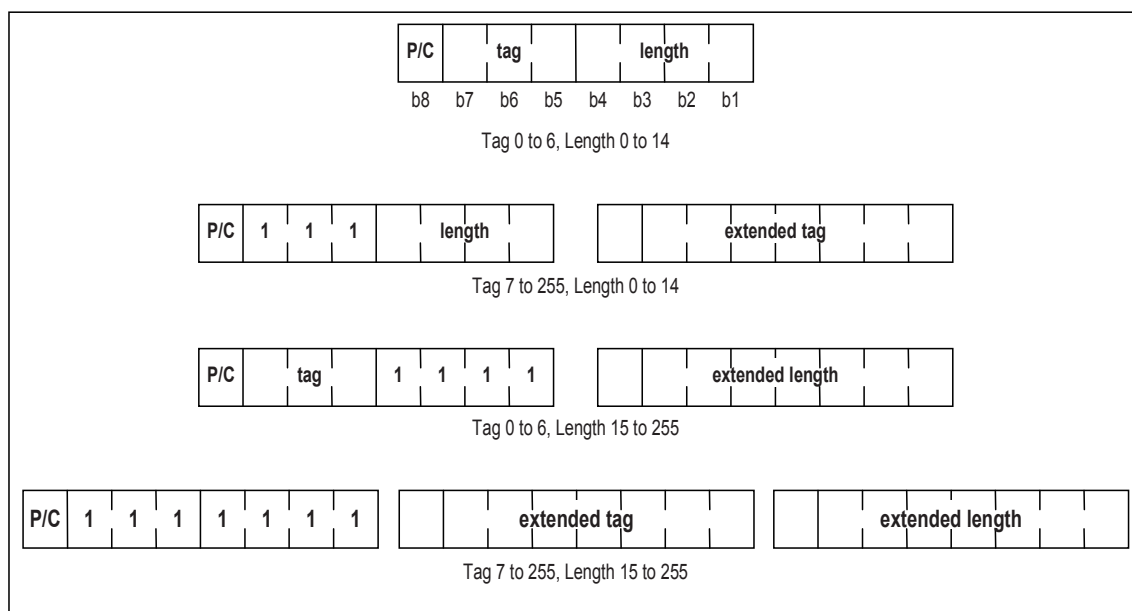


Figure 2 – Identification

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

P/C Flag =0 <=> simple component

P/C Flag =1 <=> structured component

The tag identifies the semantics of the component.

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

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

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

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

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

5.1.3.2 Data types

5.1.3.2.1 General

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



Figure 3 – Coding with identification

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



Figure 4 – Coding without identification

5.1.3.2.2 Coding for Boolean type

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

Notation: Range of values:		Boolean							
Coding:		true or false false is represented by the value 0, true is represented by the value FF							
bits		8	7	6	5	4	3	2	1
octet									
1		1	1	1	1	1	1	1	1

Figure 5 – Representation of the value true