BS EN 61158-6-5:2014

BSI Standards Publication

Industrial communication networks — Fieldbus specifications

Part 6-5: Application layer protocol specification — Type 5 elements

bsi.

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 61158-6-5:2014. It is identical to IEC 61158-6-5:2014. It supersedes BS EN 61158-6-5:2008 which is withdrawn.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.

ISBN 978 0 580 79471 1
ICS 25.040.40; 35.100.70; 35.110

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 November 2014.

Amendments issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 61158-6-5

October 2014

ICS 25.040.40; 35.100.70; 35.110

Supersedes EN 61158-6-5:2008

English Version

Industrial communication networks - Fieldbus specifications - Part 6-5: Application layer protocol specification - Type 5 elements (IEC 61158-6-5:2014)

Réseaux de communication industriels - Spécifications des bus de terrain - Partie 6-5: Spécification du protocole de la couche application - Éléments de type 5 (CEI 61158-6-5:2014)

Industrielle Kommunikationsnetze - Feldbusse - Teil 6-5: Protokollspezifikation des Application Layer (Anwendungsschicht) - Typ 5-Elemente (IEC 61158-6-5:2014)

This European Standard was approved by CENELEC on 2014-09-23. 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 CEN-CENELEC Management Centre 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 CEN-CENELEC Management Centre has the same status as the official versions.

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

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

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Foreword

The text of document 65C/764/FDIS, future edition 2 of IEC 61158-6-5, 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 approved by CENELEC as EN 61158-6-5:2014.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2015-06-23
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2017-09-23

This document supersedes EN 61158-6-5:2008.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

This document has been prepared under a mandate given to CENELEC by the European Commission and the European Free Trade Association.

Endorsement notice

The text of the International Standard IEC 61158-6-5:2014 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC 61158-3-1	NOTE	Harmonized as EN 61158-3-1.
IEC 61158-4-1	NOTE	Harmonized as EN 61158-4-1.
IEC 61784-1	NOTE	Harmonized as EN 61784-1.
IEC 61784-2	NOTE	Harmonized as EN 61784-2.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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

NOTE 1 When an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu.

Publication	Year	Title	EN/HD	Year
IEC 61158-1	-	Industrial communication networks - Fieldbus specifications - Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series	EN 61158-1	-
IEC 61158-5-5	-	Industrial communication networks - Fieldbus specifications - Part 5-5: Application layer service definition - Type 5 elements	EN 61158-5-5	-
ISO/IEC 7498-1	-	Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model	-	-
ISO/IEC 8825	1990	Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)	-	-
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	-	-
IETF RFC 791	-	Internet Protocol	-	-

CONTENTS

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 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 6-5: Application layer protocol specification – Type 5 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 5 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 5 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 5 IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI Application Layer Structure (ISO/IEC 9545).

FAL services and protocols are provided by FAL application-entities (AE) contained within the application processes. The FAL AE is composed of a set of object-oriented Application Service Elements (ASEs) and a Layer Management Entity (LME) that manages the AE. The ASEs provide communication services that operate on a set of related application process object (APO) classes. One of the FAL ASEs is a management ASE that provides a common set of services for the management of the instances of FAL classes.

Although these services specify, from the perspective of applications, how request and responses are issued and delivered, they do not include a specification of what the requesting and responding applications are to do with them. That is, the behavioral aspects of the applications are not specified; only a definition of what requests and responses they can send/receive is specified. This permits greater flexibility to the FAL users in standardizing such object behavior. In addition to these services, some supporting services are also defined in this standard to provide access to the FAL to control certain aspects of its operation.

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 61158-5-5.

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 series.

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 documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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 61158-1, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

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

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

ISO/IEC 8825:1990, *Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*[1]

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*

IETF RFC 791, *Internet Protocol;* available at <http://www.ietf.org>

[1] Withdrawn

3 Terms, definitions, symbols, abbreviations and conventions

For the purposes of this document, the following terms, definitions, symbols, abbreviations and conventions 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 61158-1 terms

For the purposes of the present document, the following IEC 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 1 to entry: 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
component that 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

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 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
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

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

3.2.9
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

Note 1 to entry: 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

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 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

Note 1 to entry: Its description includes the relationship between attribute values and services.

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

Note 1 to entry: 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
attribute that is shared by all objects within the same class

3.2.15
class code
unique identifier assigned to each object class

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

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

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

Note 1 to entry: On Dedicated ARs, only the FAL Header and the user data are transferred.

3.2.21
device
physical hardware connection to the link

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

3.2.22
device profile
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

Note 1 to entry: 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

Note 1 to entry: 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
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

Note 1 to entry: Managing includes functions such as controlling, monitoring, and diagnosing.

3.2.31
network
series of nodes connected by some type of communication medium

Note 1 to entry: 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

Note 1 to entry: 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

Note 1 to entry: 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) 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

Note 1 to entry: 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

Note 1 to entry: 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 series. 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
1)          means value of a 'reason' parameter is assigned to a parameter called 'Identifier.'
```

"xxx" Indicates fixed value.

```
Example:
    Identifier := "abc"
2)          means value "abc" is assigned to a parameter named 'Identifier.'
3)
```

=	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
4)      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
5)    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 Protocol

4.1 Overview

The Type 5 services and protocols are provided by the Type 5 AE, referred to as the *Field Device Access (FDA) Agent* throughout this specification.

The FDA Agent conveys Type 9 APDUs using Type 9 Application Relationships, referred to as *FDA Sessions*, or more simply as *sessions* throughout this Type 5 specification.

The FDA Agent may provide gateway functions that map between Type 5 services and corresponding services that operate over a Type "X" DLL. Networks that employ the Type "X" DLL are referred to as *Type "X"* networks.

4.2 FAL syntax description

4.2.1 PDU abstract syntax

The abstract syntax of APDUs is combined with their transfer syntax and is specified in 4.3.

4.2.2 Abstract syntax of data types

The abstract syntax of data types is combined with their transfer syntax and is specified in 4.3.

4.3 Transfer syntax

4.3.1 General

The transfer syntax combines the specification of the abstract syntax and their encodings as a set of fixed format APDUs. Each APDU contains a header and an APDU body. An optional trailer is also defined. The presence of the trailer is indicated in the APDU header.

The format of the APDU header and trailer is specified in 4.3.3 and 4.3.4. The APDU header contains the fields common to all messages that are mandatory. The APDU trailer contains the fields common to all APDUs that are optional.

The formats of the APDU bodies are specified in 4.3.5. If a service request or response APDU has no parameters, it is stated in the subclause that describes the APDU body. If the service has no response or error APDU body associated with it, then the corresponding subclause is not present. Common error parameters are used in many service-specific APDUs. Their definitions are specified in 4.3.6.1.

User data, when present in an APDU, is defined as a service-specific APDU body parameter. The content and length of the user data are dependent upon the service and the object being accessed. Therefore, neither is specified as part of the APDU body. However, the user data length can be calculated from the APDU length contained in the APDU header.

The APDU header, trailer, and bodies are composed of a subset of data types defined in IEC 61158-5. The encodings for these data types are defined next, followed by the definitions of the header, trailer, and APDU bodies.

4.3.2 Data type encodings

The following data types are used in APDUs as listed in Table 2. Other data types may be conveyed as user data.

Table 2 – Data types

Data Type	Octet Length	Comment
Boolean	1	0 = False, Non-Zero = TRUE
Integer8	1	
Integer16	2	
Unsigned8	1	
Unsigned16	2	
Unsigned32	4	
VisibleString	-	The length of a VisibleString is defined by the field definition.
OctetString	-	The length of an OctetString is defined by the field definition.
Time value	8	

For all data types, the bit and octet order of transmission is as defined in Type 9. The unsigned integer value for each bit position is shown in Table 3.

NOTE The resulting encoding is the same as that the encoding specified in Appendix B of RFC 791 – Internet Protocol. The bit and byte order specified in RFC 791 Appendix B is the big endian octet ordering (most significant byte first for integers) and bit ordering within bytes is most significant bit first. Although the bit numbering is opposite of that specified in FMS, both specify the high order bit (the leftmost bit) as the most significant bit. In the descriptions that follow, bit 1 is the most significant bit in each byte. The unsigned integer value for each bit position is shown in the table below.

Table 3 – Data types

Bit Position	Hex Value	Decimal Value
8	0x80	128
7	0x40	64
6	0x20	32
5	0x10	16
4	0x08	8
3	0x04	4
2	0x02	2
1	0x01	1

4.3.3 APDU header

4.3.3.1 APDU header format

The header is the same for each APDU and has a fixed length.

All header fields are mandatory.

The format of the header is shown in Table 4.

Table 4 – APDU header format

Parameter name	Octet offset	Data type	Octet length	Description
Type 5 APDU Version	0	Unsigned8	1	Specifies the Version number of the Type 5 APDU format. The version number of the Type 5 APDUs specified in this document is 1. 1 = Version 1
Options	1	Unsigned8	1	Bit 8: 1 = APDU Number present in the Trailer Bit 7: 1 = Invoke Id present in the Trailer (this bit shall be set for all client / server sessions) Bit 6: 1 = Time Stamp present in the Trailer Bit 5: Reserved, set to 0 Bit 4: 1 = Extended Control Field present in the Trailer Bit 3-1: Pad Length = Specifies the number of pad bytes that are inserted between the user data and the trailer fields. Each pad byte is set to binary zero.
ASE Id And Confirmed Msg Type	2	Unsigned8	1	Bits 8–3: ASE Id – Used to specify the ASE whose service is being conveyed by the APDU. They are interpreted as a 6-bit unsigned integer with the following values: 0 = Unused 1 = AR ASE 2 = SMK ASE 3 = Type 9 ASEs/Type 5 VFD ASE 4 = LAN Redundancy ASE 5-63 : Reserved Bits 1, 2: Confirmed Msg Type – Used to differentiate between request, response, and error APDUs. Not used with unconfirmed services. These bits are interpreted as a 2-bit unsigned integer with the following values: 0 = Request APDU 1 = Response APDU 2 = Error APDU 3 = Reserved
Service	3	Unsigned8	1	Specifies the service within the specified protocol. Bit 8: Confirmed Flag (Confirmed=1) and (Unconfirmed=0). When this flag is set for a request APDU, the corresponding response (either a response APDU or an error APDU) is always returned after the request has been processed to indicate the result of the processing. Bits 7-1 Service Id of the service (7-bit unsigned integer). The value used for each Service Id is specified in its APDU Body specification below. It is the number in parenthesis that follows the service name in the header for each service. All other values are reserved.
FDA Address	4	Unsigned32	4	The format and use of the FDA Address is dependent on the ASE Id and the type of S-VFD Context. The next subclause describes this field.
APDU Length	8	Unsigned32	4	Specifies the number of octets contained in the entire APDU, including the header and the trailer.

4.3.3.2 FDA address use

4.3.3.2.1 FDA address use summary

Subclause 4.3.3.2.1 specifies the values for the FDA Address field of the header. See Table 5 for details. In the descriptions that follow, a Type 9 device is a device with the Type 9 AL and a DLL whose addressing can be represented as a Type 1 DLL address. A Type 9 SMK is the SMK in a Type 9 device.

Table 5 – FDA address use

ASE	VCR	Bits	Use
AR	N/A	32-1	Service-specific.
LAN Redundancy	N/A	32-1	Not Used, set to 0.
SMK	N/A	32-17	Link Id Set to 0 if the destination SMK is located in the Type 5 device receiving the APDU. If the device is a Linking Device, and the destination SMK is in a Type 9 device attached to the Linking Device, then the FDA Address field contains the link id used to access the Type 9 device.
		16-1	Node.Selector The node value is the high order 8 bits and the selector value is the low order 8 bits, when taken as a16-bit unsigned integer value. When used to identify a Type 5 SMK (when the Link Id is zero), the node value is set to 0 and the selector value is set to 2. When used to identify a Type 9 SMK (when the Link Id is non-zero), this is the individual or group Type 9 node.selector address. See the description of the Find Tag Query APDU for a detailed description of its use of the FDA Address.
Type 9	Client/Server	32-17	Link Id If the server S-VFD is located in the Type 5 device, then this is 0. If the device is a Linking Device, and the destination VFD is in a Type 9 device attached to the Linking Device, then this is the Link Id of the link to which the Type 9 device is attached.
		16-1	Type 5 Selector Initiate Request: The use of this field is dependent on the Connect Option selected for the request. See the description of the Initiate APDU for the values used. Initiate Response: The Type 5 FAL AE returns the VCR Id of the initiated VCR. All Others: The Selector returned in the Initiate Response.
Type 9	Publisher / Subscriber	32-17	Link Id If the publisher VFD is located in the Type 5 device sending the APDU, then this is combined with the Selector bits to form a 32-bit flat Dlcep address. If the device is a Linking Device, and the publisher is in a Type 9 device attached to the Linking Device, then this is the Link Id of the link to which the publishing device is attached.
		16-1	Dlcep Selector If the publisher is located in the Type 5 device sending the APDU, then this is part of the 32-bit flat Dlcep address assigned to the published data. If the device is a Linking Device, and the publisher is in a Type 9 device attached to the Linking Device, then this is the Dlcep Selector of the publisher buffer.
Type 9	Report Distribution	32-17	Link Id If the report source VFD is located in the Type 5 device sending the APDU, then this is combined with the Selector bits to form a 32-bit flat DLSAP address. If the device is a Linking Device, and the report source is in a Type 9 device attached to the Linking Device, then this is the Link Id of the link to which the reporting device is attached.
		16-1	S-VFD Context Id or Dlsap Selector If the report source VFD is located in the Type 5 device sending the APDU, then this is part of the 32-bit flat Dlsap address assigned to the Report

ASE	VCR	Bits	Use
			Source VFD. If the device is a Linking Device, and the report source is in a Type 9 device attached to the Linking Device, then this is the Dlsap Selector of the report source.
NOTE The Link Id, Node, DLCEP Selector, and DLSAP Selectors are all Type 1 DLL address components.			

4.3.3.2.2 APDU sent by a client VCR endpoint

Table 6 illustrates the use of the FDA Address header field in APDUs sent by a client VCR endpoint.

Table 6 – FDA address header field APDUs sent by a client VCR endpoint

		FDA address	
Message type	**Destination AP**	**Bits**	**Value**
Type 5 Initiate Request APDU Connect Option 1	Non-MIB VFD of Type 5 device	Bits 32-17 Bits 16-1	0 (Local Type 5 Device) Generic Selector, Returned by Find Tag Query, or obtained by some other means
	Non-MIB VFD of Type 9 device connected to Linking Device	Bits 32-17 Bits 16-1	Link Id of Type 9 Link Client Type 9 VCR Index
Type 5 Initiate Request APDU Connect Option 2	MIB VFD of Type 5 device	Bits 32-17 Bits 16-1	0 (Local Type 5 Device) 0.0
	MIB VFD of Linking Device Type 9 Link Interface	Bits 32-17 Bits 16-1	Link Id of Type 9 Link 0.0
	MIB VFD of Type 9 device connected to Linking Device	Bits 32-17 Bits 16-1	Link Id of Type 9 Link Node.0
All Other Request Messages	MIB VFD of Type 5 device	Bits 32-1	FDA Address returned in Initiate Response message
	Non-MIB VFD	Bits 32-1	FDA Address returned in Initiate Response message
	MIB VFD of Linking Device Type 9 Link Interface	Bits 32-1	FDA Address returned in Initiate Response message
	MIB VFD of Type 9 device connected to Linking Device	Bits 32-1	FDA Address returned in Initiate Response message
	Non-MIB AP of Type 9 device connected to Linking Device	Bits 32-1	FDA Address returned in Initiate Response message

4.3.3.2.3 APDUs sent by a server AR VCR endpoint

Table 7 illustrates the use of the FDA Address header field in APDUs sent by a server VCR endpoint.

Table 7 – FDA address header field APDUs sent by a server VCR endpoint

Message type	Destination AP	FDA address	
		Bits	**Value**
Type 5 Initiate Response APDU Connect Option 1	Non-MIB VFD of Type 5 device	Bits 32-17 Bits 16-1	0 (Local Type 5 Device) Type 5 VCR Id[1]
	Non-MIB VFD of Type 9 device connected to Linking Device	Bits 32-17 Bits 16-1	Link Id of Type 9 Link Type 5 VCR Id[1]
Type 5 Initiate Response APDU Connect Option 2	MIB VFD of Type 5 device	Bits 32-17 Bits 16-1	0 (Local Type 5 Device) Type 5 VCR Id[1]
	MIB VFD of Linking Device Type 9 Link Interface	Bits 32-17 Bits 16-1	Link Id of Type 9 Link Interface Type 5 VCR Id[1]
	MIB VFD of Type 9 device connected to Linking Device	Bits 32-17 Bits 16-1	Link Id of Type 9 Link Type 5 VCR Id[1]
All Other Response Messages	MIB VFD of Type 5 device	Bits 32-1	Same as Request Message Same as Request Message
	Non MIB VFD of Type 5 device	Bits 32-1	Same as Request Message Same as Request Message
	MIB VFD of Linking Device Type 9 Link	Bits 32-1	Same as Request Message Same as Request Message
	MIB VFD of Type 9 device connected to Linking Device	Bits 32-1	Same as Request Message Same as Request Message
	Non-MIB VFD of Type 9 device connected to Linking Device	Bits 32-1	Same as Request Message Same as Request Message
NOTE The selector value (bits 17-32) returned is the Type 5 VCR Index of the dynamically created VCR.			

4.3.3.2.4 APDUs sent by a publisher VCR endpoint

Table 8 illustrates the use of the FDA Address in messages sent by a publisher VCR endpoint.

Table 8 – FDA address header field APDUs sent by a publisher VCR endpoint

Message type	Publisher AP	FDA address	
		Bits	**Value**
Type 9 Request Message	AP in the Type 5 device	Bits 32-1	Flat 32-bit Type 9 address assigned to the published data buffer
	VFD of Type 9 device connected through a Type 9 link of a Linking Device	Bits 32-17 Bits 16-1	Link Id of Type 9 Link Node.Selector Dlcep of the published data buffer

4.3.3.2.5 Messages sent by a report source VCR endpoint

Table 9 illustrates the use of the FDA Address in APDUs sent by a report source VCR endpoint.

Table 9 – FDA address header field APDUs sent by a report source VCR endpoint

Message type	Report source AP	FDA address	
		Bits	Value
Type 9 Request Message	AP in the Type 5 device	Bits 32-1	Flat 32-bit Type 9 address assigned to the report source VFD.
	VFD of Type 9 device connected through a	Bits 32-17	Link Id of Type 9 Link
	Type 9 link of a Linking Device	Bits 16-1	Node.Selector Dlsap of the source of the report

4.3.4 Trailer

FDA Sessions define which trailer fields are defined for the Type 9 APDUs that they convey. The use of trailer fields by SMK ASE and LAN Redundancy APDUs is service specific. See the SMK ASE and LAN Redundancy APDU definitions for their use of trailer fields.

The presence of trailer fields is negotiated for Client / Server sessions and configured for Publisher / Subscriber and Report Distribution sessions. Their presence is indicated in the Message Header Options field of the each message. The fields that are present are positioned after the user data in the order shown in Table 10.

Table 10 – APDU trailer fields

Field name	Order	Data type	Octet length	Description
APDU Number	1	Unsigned32	4	Monotonically increasing by 1, 0 based, number of the APDU within a specific session.
Invoke Id	2	Unsigned32	4	Request/response identifier used to match responses with requests. It is required for all request / response exchanges. Invoke Ids are unique within the context of an Application Relationship.
Time Stamp	3	OctetString	8	System Time at which sender created APDU (in Time value format). It can be used by receivers to calculate the one-way transfer time from the sender.
Extended Control Field	4	Unsigned32	4	Reserved For Future Use.

4.3.5 APDU Body

4.3.5.1 General

Each service has a set of APDU bodies that are conveyed in a fixed format in each APDU body. The length of the APDU body formats may vary only if the last field of the APDU body repeats or its length is variable. Variable length fields are supported only as the last field of a APDU body (before the APDU Trailer) so that their offset from the beginning of the APDU body is constant.

The number in parenthesis that follows the name of the service in each subclause header below is the value that is used for the Service Id field in the Type 5 APDU Header.

4.3.5.2 FDA Session ASE Services

4.3.5.2.1 Open Session Service (confirmed service Id = 1)

4.3.5.2.1.1 Service overview

This service is used to open a Client/Server AR between a client AR endpoint and a server AR endpoint. The source and destination address pair identifies each AR endpoint.

The Version field in the Type 5 APDU Header in the request message indicates the version of the FDA Agent that is being requested for the AR. The responder may negotiate the version down.

The Options field in the Type 5 APDU Header in the request indicates the options that are being requested for the AR. The Invoke Id bit is always set. The responder can refuse all other options. Returning 0 in the bit position representing the option indicates refusal.

If the message number option is requested, then the trailer field contains the initial value to be used by both endpoints of the session. This value is returned in the response message. If the requester does not receive a response to the request, and elects to resend the request, it increments the message number in the trailer, while keeping the invoke id the same as the original. This causes the request to be delivered to the AR endpoint created if the first request was received and a positive response was returned. If the original request was not received, or it caused a negative response to be returned (that was not received), then the new request is treated like a new request because there is no existing AR endpoint to process the message.

The FDA Address in the APDU Header is not used and is set to 0.

If the server receives an Open Session request message for a session on which it has already sent an Open Session response message, it closes the session. If not, the following applies.

The Version field in the FDA Message Header in the request message indicates the version of the FDA that is being requested for the session. The responder may negotiate the version down.

The Open Session request message conveys session attributes for validation and use by the responder. If the responder is able to support certain of these, it is permitted to negotiate them, as defined in the Request Message Parameters table below. If the requester is not prepared to operate with the negotiated attributes returned by the responder, it is free to not open the session. It may then reissue an Open Session request with different session attributes, or if using TCP, close the associated TCP connection to close the session, as appropriate.

The Options field in the Message Header in the request message indicates the options that are being requested for the session. The Invoke Id bit is always set. The responder can refuse all other options. Returning 0 in the bit position representing the option indicates refusal.

If the message number option is requested, then the trailer field contains the initial value to be used by both endpoints of the session. This value is returned in the response message. If the requester does not receive a response to the request, and elects to resend the request, it increments the message number in the trailer, while keeping the invoke id the same as the original. This causes the request to be delivered to the session endpoint created if the first request was received and a positive response was returned. This will cause the session to close. If the original request was not received, or it caused a negative response to be returned (that was not received), then the new request is treated like a new request because there is no existing session endpoint to process the message.

For the PadLength option, the requester specifies the alignment boundary for the message. This is done by setting the PadLength to the maximum number of padding bytes that can be inserted and by padding this message accordingly. If this value requests 4-or 8-byte padding and the receiver does not support padding, it may reject the request using "invalid options", or it may accept the request and return "000" in this field.

000 = No padding

011 = pad to a 4-byte boundary. The maximum number of pad bytes that can be inserted is three. Three bytes of pad are inserted into this message.

111 = pad to 8 byte boundary. The maximum number of pad bytes that can be inserted is seven. Seven bytes of pad are inserted into this message.

The server receives requests at a generic endpoint address and creates a new endpoint at a previously unused address to process each request and to return the response. The newly created endpoint is used for all subsequent APDUs sent and received on the AR.

If a request is received with invalid parameter or unsupported values, a negative response is returned with an error class of "service" and an error code of "parameter-inconsistent" for the offending parameter. This negative response also uses the additional code value of 1, and the additional description to propose an acceptable values for each of the parameters indicated below that can be used to open the session. To convey these values the 16 octets of the additional description is interpreted as 4 Unsigned32 integers instead of as a VisibleString. The location of each parameter value in the 16 octets is specified in the descriptions below.

If the responder is does not have an available Session Index for the new session, then a negative response is returned with error class = "resource" and error code = "max sessions exceeded".

If the responder is does not have the resources to support the new session, then a negative response is returned with error class = "resource" and error code = "object creation failure" or some other appropriate error code within the "resource" error class.

Successful exchange of Open Session request and response messages results in the establishment of a session that may be used to send FMS request and response messages. If UDP is used, the server returns the response from a previously unused UDP port. This port is then used to send and receive all subsequent messages for the session.

ARs opened for MIB VFD Configuration Use are referred to as configuration ARs. Only one configuration AR is allowed at a time. If a request is received to open a configuration AR and one is already open, or if there is an Type 9 VCR to any MIB VFD (Type 5 or Type 9) with update services supported already open, then the configuration AR is refused. The AR ASE returns a negative response with error class = "service" and error code = "config-access-already-open".

Opening Publisher / Subscriber and Report Distribution ARs is local, and does not result in the exchange of Open Session Service APDUs. As a result, the Open Session Request APDU parameters for these AR types are preconfigured instead of negotiated.

4.3.5.2.1.2 Request APDU parameters

Table 11 lists the request APDU parameters.

Table 11 – Request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
AR Index	0	Unsigned32	4	This parameter contains the numeric identifier of the client AR endpoint, if one. If not, 0 is used. For the response, this parameter contains the numeric identifier of the newly opened AR.
Max Buffer Size	4	Unsigned32	4	This parameter defines the maximum buffer length in octets (the buffer may contain concatenated Type 9 APDUs) that the sender of this APDU may send or receive on this AR. This parameter can be negotiated down (but not up) by the responder using the Max Buffer Size parameter. If the negotiation fails for any parameter, an acceptable value for this parameter is returned in the additional description field as an Unsigned32 at offset 0. If the requested value is supported, but another parameter failed the negotiation, then the requested value is returned at offset 0.
Max Message Length	8	Unsigned32	4	This parameter defines the maximum length in octets of APDUs to be sent on this AR by the sender of this APDU. It is used by the receiving AR endpoint as its Max Message Length attribute. This parameter cannot be negotiated. The server may reject the value supplied by the client if it cannot support it using error class = “access” and error code = “unsupported max APDU length”. If the negotiation fails for any parameter, an acceptable value for this parameter is returned in the additional description field as an Unsigned32 at offset 4. If the requested value is supported, but another parameter failed the negotiation, then the requested value is returned at offset 4.
Reserved	12	Unsigned8	1	Unused, set to 0.
Configuration Use	13	Unsigned8	1	0 = Configuration Not Permitted 1 = Configuration Permitted
Inactivity Close Time	14	Unsigned16	2	This parameter identifies how long, in seconds, that the AR stays open without receiving an APDU. After experiencing inactivity on the AR for this period of time, the AREP is closed, along with all VCR activity associated with the AREP. The responder can negotiate this value down. The value 0 is not permitted. If the negotiation fails for any parameter, an acceptable value for this parameter is returned in the additional description field as an Unsigned32 at offset 8. If the requested value is supported, but another parameter failed the negotiation, then the requested value is returned at offset 8.
Transmit Delay Time	16	Unsigned32	4	This parameter is used to set the Transmit Delay Time attribute in the receiving AREP. Its value may not be negotiated. Its value is expressed in milliseconds. If the negotiation fails for any parameter, an acceptable value for this parameter is returned in the additional description field

Parameter name	Octet offset	Data type	Octet length	Description
				as an Unsigned32 at offset 12. If the requested value is supported, but another parameter failed the negotiation, then the requested value is returned at offset 12.
SMK PD Tag	20	VisibleString	32	SMK PD Tag of the server. If the SMK PD Tag in the request message does not match the PD Tag of the server, then the server rejects the request with error class “access” and error code “object-access-denied”.

4.3.5.2.1.3 Response APDU parameters

Same as Request APDU

4.3.5.2.1.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.2.2 Idle (confirmed service Id = 3)

4.3.5.2.2.1 Service overview

This APDU is used to inform the receiver that the sender is present.

4.3.5.2.2.2 Request APDU parameters

None.

4.3.5.2.2.3 Response APDU parameters

None.

4.3.5.3 SMK ASE services

4.3.5.3.1 SM find tag query (unconfirmed service Id = 1)

4.3.5.3.1.1 Service overview

The network address used to send the FIND_TAG_QUERY message and the FDA Address in the Type APDU Header together indicate which SMKs are to receive and process the message.

The Find Tag Query message may be sent to one or more SMKs in Type 5 and/or Type 9 devices. When sent to a linking device, the linking device Type 5 SMK responds if the linking device contains the queried object. The Type 9 interface SMKs in the linking device are used only to forward the query to Type 9 links. They never respond to the queries.

Although this service is unconfirmed, it uses the Invoke Id in the Message Trailer to allow it to be matched to the returned SM Find Tag Reply message.

Table 12 describes the distribution that may be requested for this service. The FDA Addresses in the table are shown in hexidecimal.

Table 12 – SMK FDA address values

FDA address meaning	FDA address		Destination address	Distribution
	LLLL	NN.SS		
Type 5 Device SMK	00 00	00.02	SMK Multicast	All Type 5 SMKs
			Individual	Individual Type 5 SMK
Type 5 and Type 9 SMK Local Link Group Address	00 00	01.09[1]	SMK Multicast	All Type 5 SMKs and all Type 9 SMKs
			Individual	Individual LD SMK and all Type 9 SMKs accessible through the LD. This does not include the SMKs of the linking device's Type 9 interfaces
Type 9 SMK Local Link Group Address	LL LL	01.09[1]	SMK Multicast	All Type 9 SMKs on Type 9 link LL LL
			Individual	All Type 9 SMKs on Type 9 link LL LL accessible through the LD. This does not include the SMK of the linking device's Type 9 LL LL interface
Individual Type 9 Address	LL LL	NN.02	Individual	Individual Type 9 SMK accessible through the LD
NOTE Specified in IEC 61158-4-1as the Type 1 link specific address for "the SMAEs of all DLEs on the link".				

4.3.5.3.1.2 Request APDU parameters

Table 13 lists the SMK FDA address values.

Table 13 – SMK FDA address values

Parameter name	Octet offset	Data type	Octet length	Description
Query Type	0	Unsigned8	1	Indicates the type of query: 0 = PD Tag query for primary or non-redundant device 1 = VFD tag query 2 = Function-Block tag query 3 = Element Id query 4 = PD Tag/VFD Reference query 5 = Device Index query 6 = PD Tag query for secondary or member of redundant set All other values are reserved.
Reserved	1	Octetstring	3	Reserved, set to 0
Element Id Or VFD Reference	4	Unsigned32	4	Service Parameter When not used, set to binary 0.
PD Tag Or Object Tag	8	VisibleString	32	Service Parameter When not used, set to binary 0.
VFD Tag	40	VisibleString	32	Service Parameter When not used, set to binary 0.

4.3.5.3.2 SM find tag reply (unconfirmed service Id = 2)

4.3.5.3.2.1 Service overview

This message is used to reply to the SM Find Tag request message.

Although this service is unconfirmed, it uses the Invoke Id in the Type 5 APDU Trailer to allow it to be matched to the corresponding SM Find Tag Query message.

4.3.5.3.2.2 Request APDU parameters

Table 14 lists the request APDU parameters.

Table 14 – Request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Query Type	0	Unsigned8	1	Indicates the type of query: 0 = PD Tag query for primary or non-redundant device 1 = VFD tag query 2 = Function-Block tag query 3 = Element Id query 4 = PD Tag/VFD Reference query 5 = Device Index query 6 = PD Tag query for secondary or member of redundant set All other values are reserved.
Type 9 Node Address	1	Unsigned8	1	Service Parameter Set to 0 if the Reply is for a Type 5 device
Type 9 Link Id	2	Unsigned16	2	Service Parameter Set to 0 if the Reply is for a Type 5 device.
VFD Reference	4	Unsigned32	4	Service Parameter
Queried Object Numeric Id	8	Unsigned32	4	Service Parameter
Queried Object Network Address	12	Octetstring	16	Service Parameter
Queried Object OD Version Number	28	Unsigned32	4	Service Parameter
Queried Object Device ID	32	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Queried Object PD Tag	64	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Duplicate Detection State	96	Unsigned8	1	Service Parameter Encoded as follows. Bits 8-3: Reserved, set to 0. Bit 2: 1 = Duplicate PD Tag Detected 0 = Duplicate PD Tag Not Detected Bit 1: 1 = Duplicate Device Index Detected 0 = Duplicate Device Index Not Detected
Reserved	97	Unsigned8	1	Reserved, set to 0.
Number in List of FDA Addresses	98	Unsigned16	2	This parameter indicates how many FDA Address Selectors are contained in the List of FDA Address Selectors parameter. This value of this attribute is 0 if the Query Type is “physical-device query”.

Parameter name	Octet offset	Data type	Octet length	Description
List of FDA Addresses	100	Unsigned16	2x number in list	Service Parameter Each Selector is an Unsigned32 that immediately follows its predecessor in the series (no spaces or padding between them).

4.3.5.3.3 SM Identify (confirmed service Id = 3)

4.3.5.3.3.1 Service overview

This service is used to request the identity of a single Type 5 or Type 9 device or of a group of Type 5 devices.

Table 15 specifies the network addresses and FDA Addresses used with the SM Identify request message.

Table 15 – SMK FDA address values for SM identify

Identification of	FDA address LLLL.NN.SS	Network address	Request message delivered to
Type 5 SMK device, including	00 00.00.02	SM Multicast	All Type 5 SMKs
Live List Version Number List[a]		Individual	Individual Type 5 SMK
Linking Device Node Version Number List for Specific Type 9 Interface	LL LL.00.02	SM Multicast	All Linking Device Type 5 SMKs – the one with the requested Link Id responds
	LL LL.00.02	Individual	Linking Device Type 5 SMK
Type 9 device SMK	LL LL.NN.02	Individual	Linking Device Type 5 SMK -> Type 9 SMK

[a] The Live List Version Number List is returned by Linking Device SMKs.

4.3.5.3.3.2 Request APDU parameters

None.

4.3.5.3.3.3 Response APDU parameters

Same as SM Device Annunciation Request APDU parameters.

4.3.5.3.3.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.3.4 SMK clear address (confirmed service Id = 12)

4.3.5.3.4.1 Service overview

This service is always confirmed and, therefore, always uses the Invoke Id in the Message Trailer.

Table 16 specifies the Network Address and FDA Address used with the SM Clear Address request and response APDUs.

Table 16 – SMK FDA address values for SMK set assignment info request APDUs

Use	FDA address LLLL.NN.SS	Network address	Request path
Type 5 Clear Address	00 00.00.02	Individual	Type 5 Requester -> Type 5 SMK
Type 9 Clear Address	LL LL.NN.02	Individual	Type 5 Requester -> LD FDA Agent -> Type 9 SMK

4.3.5.3.4.2 Request APDU parameters

Table 17 specifies the request APDU parameters.

Table 17 – SMK clear address request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Device Id	0	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Physical Device Tag	32	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
InterfaceToClear	64	Unsigned8	1	Service Parameter Encoded as follows: 0 = Interface A 1 = Interface B 255 = All interfaces
Reserved	65	Octetstring	3	Reserved, set to 0.

4.3.5.3.4.3 Response APDU parameters

None.

4.3.5.3.4.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.3.5 SMK set assignment info (confirmed service Id = 14)

4.3.5.3.5.1 Service overview

This service is always confirmed and, therefore, always uses the Invoke Id in the Message Trailer.

Table 18 specifies the Network Address and FDA Address used with the SM Set Assignment Info request and response APDUs.

Table 18 – SMK FDA address values for SMK set assignment info request APDUs

Use	FDA address LLLL.NN.SS	Network address	Request path
Type 5 Set Assignment Info	00 00.00.02	Individual	Type 5 Requester -> Type 5 SMK
Type 9 Set Assignment Info	LL LL.NN.02	Individual	Type 5 Requester -> LD FDA Agent -> Type 9 SMK

4.3.5.3.5.2 Request APDU parameters

Table 19 specifies the request APDU parameters.

Table 19 – SMK set assignment info request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Device Id	0	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Physical Device Tag	32	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Type 9 New Address	64	Unsigned8	1	Service Parameter The field contains the new Type 9 address for the Type 9 device identified in the FDA Address of the Type 5 APDU Header. The value of this field is 0 if the FDA Address does not identify a Type 9 device or this APDU is not being used to change the Type 9 device's address.
Device Redundancy State	65	Unsigned8	2	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device. The value 0 indicates that the device is not participating in device redundancy. If it is to participate in device redundancy, the value of bit pair 2 and 1 is non-zero. The value of bit pair 4 and 3 is non-zero whenever bits 2 & 1 indicate "External Switchover Control Device". When the bit pair 1 and 2 indicate "External Switchover Control Device", bits 3 and 4 must be set to “2” to indicate that the current primary is to become the secondary, and one of the secondaries is to become the primary. Bit 8 = Reserved, set to 0 Bit 7 = Reserved, set to 0 Bit 6 = Reserved, set to 0 Bit 5 = Reserved, set to 0 Bits 4 & 3 –External Switchover Control Device Redundancy Role 0 = Not used 1 = Primary 2 = Secondary Bits 2 & 1 – Assigned Device Redundancy Type 0 = Non-redundant 1 = External Switchover Control 2 = Internal Switchover Control
LAN Redundancy Socket Address	66	Unsigned16	2	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device. Set to the Reserved LAN Redundancy port if the FDA Address identifies Type 5 device.
Annunciation Repeat Time	68	Unsigned32	4	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device.
Device Index	72	Unsigned16	2	Service Parameter Encoded as follows.

Parameter name	Octet offset	Data type	Octet length	Description
				0 = Index not assigned 1 to Max Device Index = Device Index
Max Device Index	74	Unsigned16	2	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device.
Operational Network Address	76	Octetstring	16	Service Parameter Unused and set to 0 if the FDA Address identifies an Type “X” device.
Reserved	92	Octetstring	3	Reserved, set to 0.
Clear Duplicate Detection State	95	Unsigned8	2	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device. Encoded as follows: Bits 8-3: Reserved, set to 0. Bit 2: 1 = Do not clear Duplicate PD Tag Detected 0 = Clear Duplicate PD Tag Detected Bit 1: 1 = Do not clear Duplicate Device Index Detected 0 = Clear Duplicate Device Index Detected

4.3.5.3.5.3 Response APDU parameters

Table 20 specifies the response APDU parameters.

Table 20 – SMK set assignment info response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Reserved	0	Unsigned16	2	Reserved, set to 0.
Max Device Index	2	Unsigned16	2	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device.
Annunciation Repeat Time	4	Unsigned32	4	Service Parameter Unused and set to 0 if the FDA Address identifies a Type 9 device.

4.3.5.3.5.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.3.6 SMK clear assignment info (confirmed service Id = 15)

4.3.5.3.6.1 Service overview

This service is always confirmed and, therefore, always uses the Invoke Id in the Message Trailer.

Table 21 specifies the Network Address and FDA Address used with the SM Clear Assignment Info request and response APDUs.

Table 21 – SMK FDA address values for SMK device clear assignment Info APDUs

Use	FDA address LLLL.NN.SS	Network address	Request path
Type 5 Clear Assignment Info	00 00.00.02	Individual	Type 5 Requester -> Type 5 SMK
Type 9 Clear Assignment Info	LL LL.NN.02	Individual	Type 5 Requester -> LD FDA Agent -> Type 9 SMK

4.3.5.3.6.2 Request APDU parameters

Table 22 defines the request APDU parameters.

Table 22 – SMK clear assignment info request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Device Id	0	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Physical Device Tag	32	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.

4.3.5.3.6.3 Response APDU parameters

None.

4.3.5.3.6.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.3.7 SMK device annunciation (unconfirmed service Id = 16)

4.3.5.3.7.1 Service overview

The request APDU body format defined below is used for the SM Device Annunciation service and also for the SM Identify service.

This service is always unconfirmed and no APDU Trailer fields are used.

Table 23 specifies the Network Address and FDA Address used with the SMK Device Annunciation request APDUs.

Table 23 – SMK FDA address values for SMK device annunciation request APDUs

Use	FDA address LLLL.NN.SS	Network address	Request path
Device Annunciation	00 00.00.02	SM Multicast	Type 5 SMK -> Configuration Applications

4.3.5.3.7.2 Request APDU parameters

Table 24 defines the request APDU parameters.

Table 24 – SMK device annunciation request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
SMK State	0	Unsigned8	1	Service Parameter Encoded as follows. Bits 2-8: 0 = Reserved 1 = NO_TAG 2 = OPERATIONAL 3-127 = Reserved Bit 1: 0 = Not Synchronized with SNTP Time Server 1 = Synchronized with SNTP Time Server
Device Type	1	Unsigned8	1	Service Parameter Encoded as follows. Each bit identifies a different device type. It is the destination device type for request messages and the source device for response messages. A combination of bits 8-6 and 2,1 may be set for devices. Bit 5 is used only for Type “X” devices, and only in SM Identify responses. Non Devices participating in device assignment set the value of bits 8 through 3 of this parameter to 0. Bit 8 = Linking Device Bit 7 = I/O Gateway Bit 6 = Field Device Bit 5 = Type “X” Device Bit 4 = Reserved Bits 3-1 – Redundant Device Type Capability 0 = Non-redundant 1 = External Switchover Control 2 = Internal Switchover Control 3 = External and Internal Switchover Control 4 = Not used 5 = Internal Switchover Control and Non-redundant device 6 = External Switchover Control and Non-redundant device 7 = External and Internal Switchover Control and Non-redundant device
Device Redundancy State	2	Unsigned8	2	Service Parameter Encoded as follows. Encoded as follows. Bit 8 = Reserved, set to 0 Bit 7 = Reserved, set to 0 Bit 6 = Reserved, set to 0 Bit 5 = Reserved, set to 0 Bits 4 & 3 – Device Redundancy Role 0 = Non-redundant 1 = Primary 2 = Secondary Bits 2 & 1 – Assigned Redundant Device Type 0 = Non-redundant

Parameter name	Octet offset	Data type	Octet length	Description
				1 = External Switchover Control 2 = Internal Switchover Control
Duplicate Detection State	3	Unsigned8	2	Service Parameter Encoded as follows. Bits 8-3: Reserved, set to 0. Bit 2: 1 = Duplicate PD Tag Detected 0 = Duplicate PD Tag Not Detected Bit 1: 1 = Duplicate Device Index Detected 0 = Duplicate Device Index Not Detected
Device Index	4	Unsigned16	2	Service Parameter Encoded as follows. 0 = Index not assigned 1 to Max Device Index = Device Index
Max Device Index	6	Unsigned16	2	Service Parameter
Operational Network Address	8	Octetstring	16	Service Parameter
Device Id	24	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Physical Device Tag	56	VisibleString	32	Service Parameter Unused octets in this field are set to ASCII space characters.
Annunciation Repeat Time	92	Unsigned32	4	Service Parameter
LAN Redundancy Socket Address	96	Unsigned16	2	Service Parameter
Reserved	98	Unsigned16	2	Service Parameter
APDU Version Number	100	Unsigned32	4	Service Parameter
Device Version Number	100	Unsigned32	4	Service Parameter
Number of Entries in Version Number List	104	Unsigned32	4	This parameter indicates how many version numbers are contained in the Version Number List field below. Its value is 0 if the device is not a linking device or a gateway.
Version Number List	108	Unsigned32	4 x Number of Entries	Service Parameter This field is present only for linking devices and I/O gateways. This field contains a list of numbers. Each is an Unsigned32 number. The number of entries in the list is specified by the field above "Number of Entries in Version Number List". This field supports two types of lists, as determined by the Link Id portion of the FDA Address, as follows: Link Id = 0 For linking devices, this is a list of Type 9 live list version numbers, one for each Type 9 interface attached to the linking device. Each element in the list is constructed as follows: Bits 32-17: Type 9 Link Id Bits 16-9: Reserved, set to 0

Parameter name	Octet offset	Data type	Octet length	Description
				Bits 8-1: Version Number The list is ordered by Type 9 bridge interface number with the Live List Version for the first interface occurring first in the list. The value 0 is used for the Link Id bits if the interface has not been assigned a link Id. Link Id > 0 This is a List of Type “X” Node Address Version Numbers, one for each node id currently in the live list for the specified Type “X” link, including the LD Type “X” interface. The List of Type “X” Node Address Version Numbers is used only when this message format is used for SM Identify Responses that describe an LD Type “X” interface. The list begins with the version number of the specified Type “X” Interface. The remainder of the list is sorted in ascending order by Type “X” node address, with two version numbers packed into each Unsigned32 value, as follows: Bits 33-25: Type “X” Node Address Bits 24-17 Version Number Bits 16-9: Type “X” Node Address Bits 8-1: Version Number

4.3.5.4 Type 9 ASE Services

4.3.5.4.1 General note

NOTE Unless specified otherwise, all parameters of the Type 9 Service APDUs and their values are defined in IEC 61158-5 subseries.

4.3.5.4.2 Reject

The Type 9 Reject Service is used only to indicate to the requester that the response is too long for Type 9 to convey it. Instead of defining a separate APDU here to convey this information, the Common Error APDU conveys it, using the Error Class "Reject" and the Reject Code as defined in the Type 9 abstract syntax for the Error Code.

4.3.5.4.3 Initiate (confirmed service Id = 96)

4.3.5.4.3.1 Request APDU parameters

Table 25 defines the request APDU parameters.

Table 25 – Initiate request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Connect Option	0	Unsigned8	1	This field indicates which option is to be used to open the S-VFD Context. Its values are: 1 = VCR Selector 2 = MIB Access 3 = FBAP Access
Access Protection Supported Calling	8	Boolean	1	Service Parameter
Password and Access Groups Calling	2	Unsigned16	2	Service Parameter
Version OD Calling	4	Integer16	2	Service Parameter

Parameter name	Octet offset	Data type	Octet length	Description
Profile Number Calling	6	Unsigned16	2	Service Parameter
PD Tag	8	OctetString	32	The SMK Physical Device Tag of the device containing the destination VFD.

4.3.5.4.3.2 Response APDU parameters

Table 26 lists the response APDU parameters.

Table 26 – Initiate response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Version OD Called	0	Integer16	2	Service Parameter
Profile Number Called	2	Unsigned16	2	Service Parameter

4.3.5.4.3.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1. See Type 9 Initiate specific error codes.

4.3.5.4.4 Abort (unconfirmed service Id = 112)

4.3.5.4.4.1 Request APDU parameters

Table 27 specifies the request APDU parameters.

Table 27 – Abort request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Abort Detail	0	OctetString	16	Supplied by the sender of the APDU.
Abort Identifier	16	Unsigned8	1	Service parameter. Values defined for Type 9 PDUs.
Reason Code	17	Unsigned8	1	Service Parameter
Reserved	18	Unsigned16	2	Reserved, set to 0

4.3.5.4.5 Get status (confirmed service Id = 0)

4.3.5.4.5.1 Request APDU parameters

None.

4.3.5.4.5.2 Response APDU parameters

Table 28 defines the response APDU parameters.

Table 28 – Get response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Logical Status	0	Unsigned8	1	Service Parameter
Physical Status	1	Unsigned8	1	Service Parameter
Reserved	2	Unsigned16	2	Reserved, set to 0
Local Detail	4	OctetString	4	Service Parameter. The high order octet is reserved and set to 0. The low order three octets contain the 3-octet Local Detail value

4.3.5.4.6 Status notification (unconfirmed service Id = 1)

4.3.5.4.6.1 Request APDU parameters

Same as Get Status Response APDU

4.3.5.4.7 Identify (confirmed service Id = 1)

4.3.5.4.7.1 Request APDU parameters

None.

4.3.5.4.7.2 Response APDU parameters

Table 29 specifies the response APDU parameters.

Table 29 – Identify response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Vendor Name	0	VisibleString	32	Service Parameter
Model Identifier	32	VisibleString	32	Service Parameter
Vendor Revision	64	VisibleString	32	Service Parameter

4.3.5.4.8 Get OD (confirmed service Id = 4)

4.3.5.4.8.1 Request APDU parameters

Table 30 specifies the request APDU parameters.

Table 30 – Get OD request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
All Attributes	0	Boolean	1	Service Parameter
Start Index Flag	1	Boolean	1	Service Parameter
Reserved	2	Unsigned16	2	Reserved, set to 0
Index	4	Unsigned32	4	Service Parameter

4.3.5.4.8.2 Response APDU parameters

Table 31 specifies the response APDU parameters.

Table 31 – Get OD response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
More Follows	0	Boolean	1	Service Parameter
Number of Object Descriptions	1	Unsigned8	1	Service Parameter
Reserved	2	Unsigned16	2	Reserved, set to 0
List of Object Descriptions	4	OctetString	N	Defined in 4.3.5.6.2

4.3.5.4.8.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.9 Initiate Put OD (confirmed service Id = 28)

4.3.5.4.9.1 Request APDU parameters

Table 32 specifies the request APDU parameters.

Table 32 – Initiate put OD request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Reserved	0	Unsigned16	2	Reserved, set to 0
Consequence	2	Unsigned16	2	Service Parameter. Values defined for Type 9 PDUs

4.3.5.4.9.2 Response APDU parameters

None

4.3.5.4.9.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.10 Put OD (confirmed service Id = 29)

4.3.5.4.10.1 Request APDU parameters

Table 33 specifies the request APDU parameters.

Table 33 – Put OD request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Number of Object Descriptions	0	Unsigned8	1	Number of entries in List of Object Descriptions
Reserved	1	OctetString	3	Reserved, set to 0
List of Object Descriptions	4	OctetString	N	Defined in 4.3.5.6.2

4.3.5.4.10.2 Response APDU parameters

None

4.3.5.4.10.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.11 Terminate put OD (confirmed service Id = 30)

4.3.5.4.11.1 Request APDU parameters

None.

4.3.5.4.11.2 Response APDU parameters

None.

4.3.5.4.11.3 Error APDU parameters

Defined by the OD Error APDU parameters in 8.3.6.

4.3.5.4.12 Generic initiate download sequence (confirmed service Id = 31)

4.3.5.4.12.1 Service overview

This service corresponds to the Type 9 Initiate Load Service with the following parameter settings:

- Load Region Index Called: Load region of the responder (Server)
- Load Type: DOWNLOAD
- Load Service to Use: Push Segment (Generic Download Segment)
- Load Service Initiator: TRUE (Initiator (Client) initiates the load service (The Push Segment service)).

4.3.5.4.12.2 Request APDU parameters

Table 34 specifies the request APDU parameters.

Table 34 – Generic initiate download sequence request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.12.3 Response APDU parameters

None

4.3.5.4.12.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.13 Generic download segment (confirmed service Id = 32)

4.3.5.4.13.1 Service overview

This service corresponds to the Type 9 Generic Download Segment Service.

4.3.5.4.13.2 Request APDU parameters

See Table 35.

Table 35 – Generic download segment request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
More Follows	4	Boolean	1	Service Parameter
Reserved	5	OctetString	3	Set to 0.
Load Data	8	OctetString	N	Service Parameter

4.3.5.4.13.3 Response APDU parameters

None.

4.3.5.4.13.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.14 Generic terminate download sequence (confirmed service Id = 33)

4.3.5.4.14.1 Service overview

This service corresponds to the Type 9 Generic Terminate Download Sequence Service.

4.3.5.4.14.2 Request APDU parameters

See Table 36.

Table 36 – Generic terminate download sequence request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.14.3 Response APDU parameters

See Table 37.

Table 37 – Response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Terminate Reason	0	Boolean	1	Service Parameter
Reserved	1	OctetString	3	Reserved, set to 0

4.3.5.4.14.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.15 Initiate download sequence (confirmed service Id = 9)

4.3.5.4.15.1 Service overview

This service corresponds to the Type 9 Initiate Download Sequence Service with the following parameter settings:

- Load Region Index Called: Load region of the responder (Server)
- Load Type: DOWNLOAD
- Load Service to Use: Pull Segment (Download Segment)
- Load Service Initiator: FALSE (Remote AP (Server) initiates the load service (The Pull Segment service)).

4.3.5.4.15.2 Request APDU parameters

See Table 38.

Table 38 – Initiate download sequence request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.15.3 Response APDU parameters

None.

4.3.5.4.15.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.16 Download segment (confirmed service Id = 10)

4.3.5.4.16.1 Service overview

This service corresponds to the Type 9 Download Segment Service.

4.3.5.4.16.2 Request APDU parameters

See Table 39.

Table 39 – Download segment request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.16.3 Response APDU parameters

See Table 40.

Table 40 – Download segment response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
More Follows	0	Boolean	1	Service Parameter
Reserved	2	OctetString	3	Reserved, each octet is set to binary 0
Load Data	4	OctetString	N	Service Parameter

4.3.5.4.16.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.17 Terminate download sequence (confirmed service Id = 11)

4.3.5.4.17.1 Service overview

This service corresponds to the Type 9 Terminate Download Sequence Service.

4.3.5.4.17.2 Request APDU parameters

See Table 41.

Table 41 – Terminate download sequence request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Reserved	4	Octetstring	3	Reserved, each octet is set to binary 0
Terminate Reason	7	Boolean	1	Service Parameter

4.3.5.4.17.3 Response APDU parameters

None.

4.3.5.4.17.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.18 Initiate upload sequence (confirmed service Id = 12)

4.3.5.4.18.1 Service overview

This service corresponds to the Type 9 Initiate Upload Sequence Service with the following parameter settings:

- Load Region Index Called: Load region of the responder (Server)
- Load Type: UPLOAD
- Load Service to Use: Pull Segment (Upload Segment)
- Load Service Initiator: TRUE (Initiator (Client) initiates the load service (The Pull Segment service))

4.3.5.4.18.2 Request APDU parameters

See Table 42.

Table 42 – Initiate upload sequence request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.18.3 Response APDU parameters

None.

4.3.5.4.18.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.19 Upload segment (confirmed service Id = 13)

4.3.5.4.19.1 Service overview

This service corresponds to the Type 9 Upload Segment Service.

4.3.5.4.19.2 Request APDU parameters

See Table 43.

Table 43 – Upload segment request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.19.3 Response APDU parameters

See Table 44.

Table 44 – Upload segment response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
More Follows	0	Boolean	1	Service Parameter
Reserved	2	OctetString	3	Reserved, each octet is set to binary 0
Load Data	4	OctetString	N	Service Parameter

4.3.5.4.19.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.20 Terminate upload sequence (confirmed service Id = 14)

4.3.5.4.20.1 Service overview

This service corresponds to the Type 9 Terminate Upload Sequence Service.

4.3.5.4.20.2 Request APDU parameters

See Table 45.

Table 45 – Terminate upload sequence request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.20.3 Response APDU parameters

None

4.3.5.4.20.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.21 Request domain download (confirmed service Id = 15)

4.3.5.4.21.1 Service overview

This service corresponds to the Type 9 Request Domain Download Service.

4.3.5.4.21.2 Request APDU parameters

See Table 46 for definitions.

Table 46 – Request domain download request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Additional Information	4	VisibleString	N	Service Parameter

4.3.5.4.21.3 Response APDU parameters

None.

4.3.5.4.21.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.22 Request domain upload (confirmed service Id = 16)

4.3.5.4.22.1 Service overview

This service corresponds to the Type 9 Create Program Invocation Service.

4.3.5.4.22.2 Request APDU parameters

See Table 47 for definitions.

Table 47 – Request domain upload request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Additional Information	4	VisibleString	N	Service Parameter

4.3.5.4.22.3 Response APDU parameters

None.

4.3.5.4.22.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.23 Create program invocation (confirmed service Id = 17)

4.3.5.4.23.1 Service overview

The Create Program Invocation corresponds to to the Type 9 Create Program Invocation service.

4.3.5.4.23.2 Request APDU parameters

See Table 48 for definitions.

Table 48 – Create program invocation request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Reusable	0	Boolean	1	Service Parameter
Reserved	1	Unsigned8	1	Reserved, set to 0
Number of Domain Indexes	2	Unsigned16	2	The number of domain indexes in this service
List Of Domain Indexes	4	Unsigned32	4 X Number of Domain Indexes	Service Parameter

4.3.5.4.23.3 Response APDU parameters

See Table 49 for definitions.

Table 49 – Create program invocation response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.23.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.24 Delete program invocation (confirmed service Id = 18)

4.3.5.4.24.1 Service overview

The Delete Program Invocation corresponds to to the Type 9 Delete Program Invocation service.

4.3.5.4.24.2 Request APDU parameters

See Table 50 for definitions.

Table 50 – Delete program invocation request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.24.3 Response APDU parameters

None.

4.3.5.4.24.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.25 Start (confirmed service Id = 19)

4.3.5.4.25.1 Request APDU parameters

See Table 51 for definitions.

Table 51 – Start request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Execution Argument	4	OctetString	N	Service Parameter

4.3.5.4.25.2 Response APDU parameters

None.

4.3.5.4.25.3 PI Error APDU parameters

Defined by the Common PI Error APDU parameters in 8.3.6.2.

4.3.5.4.26 Stop (confirmed service Id = 20)

4.3.5.4.26.1 Request APDU parameters

See Table 52 for definitions.

Table 52 – Stop request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.26.2 Response APDU parameters

None.

4.3.5.4.26.3 Error APDU parameters

Defined by the Common PI Error APDU parameters in 8.3.6.2.

4.3.5.4.27 Resume (confirmed service Id = 21)

4.3.5.4.27.1 Request APDU parameters

See Table 53 for definitions.

Table 53 – Resume request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Execution Argument	4	OctetString	N	Service Parameter

4.3.5.4.27.2 Response APDU parameters

None.

4.3.5.4.27.3 Error APDU parameters

Defined by the Common PI Error APDU parameters in 8.3.6.2.

4.3.5.4.28 Reset (confirmed service Id = 22)

4.3.5.4.28.1 Request APDU parameters

See Table 54 for definitions.

Table 54 – Reset request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.28.2 Response APDU parameters

None.

4.3.5.4.28.3 Error APDU parameters

Defined by the Common PI Error APDU parameters in 8.3.6.2.

4.3.5.4.29 Kill (confirmed service Id = 23)

4.3.5.4.29.1 Request APDU parameters

See Table 55 for definitions.

Table 55 – Kill request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.29.2 Response APDU parameters

None.

4.3.5.4.29.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.30 Read (confirmed service Id = 2)

4.3.5.4.30.1 Request APDU parameters

See Table 56 for definitions.

Table 56 – Read request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.30.2 Response APDU parameters

See Table 57 for definitions.

Table 57 – Read response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Value	0	OctetString	N	Service Parameter

4.3.5.4.30.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.31 Read with subindex (confirmed service Id = 82)

4.3.5.4.31.1 Service overview

This service corresponds to the Type 9 Read service.

4.3.5.4.31.2 Request APDU parameters

See Table 58 for definitions.

Table 58 – Read with subindex request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Component ID	4	Unsigned32	4	Service Parameter

4.3.5.4.31.3 Response APDU parameters

See Table 59 for definitions.

Table 59 – Read with subindex response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Value	0	OctetString	N	Service Parameter

4.3.5.4.31.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.32 Write (confirmed service Id = 3)

4.3.5.4.32.1 Request APDU parameters

See Table 60 for definitions.

Table 60 – Write request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Value	4	OctetString	N	Service Parameter

4.3.5.4.32.2 Response APDU parameters

None.

4.3.5.4.32.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.33 Write with subindex (confirmed service Id = 83)

4.3.5.4.33.1 Service overview

This service corresponds to the Type 9 Write service.

4.3.5.4.33.2 Request APDU parameters

See Table 61 for definitions.

Table 61 – Write with subindex request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Component ID	4	Unsigned32	4	Service Parameter
Value	8	OctetString	N	Service Parameter

4.3.5.4.33.3 Response APDU parameters

None.

4.3.5.4.33.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.34 Define variable list (confirmed service Id = 7)

4.3.5.4.34.1 Service overview

This service corresponds to the Type 9 Define Variable List service.

4.3.5.4.34.2 Request APDU parameters

See Table 62 for definitions.

Table 62 – Define variable list request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Number of Indexes	0	Unsigned32	4	Number of Variable Indexes in the list that follows.
List Of Indexes	4	Unsigned32	4 x Number of Indexes	Service Parameter

4.3.5.4.34.3 Response APDU parameters

See Table 63 for definitions.

Table 63 – Define variable list response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.34.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.35 Delete variable list (confirmed service Id = 8)

4.3.5.4.35.1 Service overview

This service corresponds to the Type 9 Delete Variable List service.

4.3.5.4.35.2 Request APDU parameters

See Table 64 for definitions.

Table 64 – Delete variable list request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter

4.3.5.4.35.3 Response APDU parameters

None.

4.3.5.4.35.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.36 Information report (unconfirmed service Id = 0)

4.3.5.4.36.1 Request APDU parameters

See Table 65 for definitions.

Table 65 – Information report request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Value	4	OctetString	N	Service Parameter

4.3.5.4.37 Information report with subindex (unconfirmed service Id = 16)

4.3.5.4.37.1 Service overview

This service corresponds on the Type 9 Information Report service.

4.3.5.4.37.2 Request APDU parameters

See Table 66 for definitions.

Table 66 – Information report with subindex request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Component ID	4	Unsigned32	4	Service Parameter
Value	8	OctetString	N	Service Parameter

4.3.5.4.38 Information report on change (unconfirmed service Id = 17)

4.3.5.4.38.1 Service overview

This service corresponds to the Type 9 Information Report service.

This APDU is used to send data that is published only when its value has changed. The VCR that controls the publishing indicates if this service is to be used.

4.3.5.4.38.2 Request APDU parameters

See Table 67 for definitions.

Table 67 – Information report on change request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Value	4	OctetString	N	Service Parameter

4.3.5.4.39 Information report on change with subindex (unconfirmed service Id = 18)

4.3.5.4.39.1 Service overview

This service corresponds to the Type 9 Information Report service.

This APDU is used to send data that is published only when its value has changed. The VCR that controls the publishing indicates if this service is to be used.

4.3.5.4.39.2 Request APDU parameters

See Table 68 for definitions.

Table 68 – Information report on change with subindex request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Component ID	4	Unsigned32	4	Service Parameter
Value	8	OctetString	N	Service Parameter

4.3.5.4.40 Event notification (unconfirmed service Id = 2)

4.3.5.4.40.1 Request APDU parameters

See Table 69 for definitions.

Table 69 – Event notification request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Event Number	4	Unsigned32	4	Service Parameter
Data	8	OctetString	N	Service Parameter

4.3.5.4.41 Alter event condition monitoring (confirmed service Id = 24)

4.3.5.4.41.1 Request APDU parameters

See Table 70 for definitions.

Table 70 – Alter event condition monitoring request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Reserved	4	Octetstring	3	Reserved, each octet is set to binary 0
Enabled	7	Boolean	1	Service Parameter

4.3.5.4.41.2 Response APDU parameters

None.

4.3.5.4.41.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.4.42 Acknowledge event notification (confirmed service Id = 25)

4.3.5.4.42.1 Request APDU parameters

See Table 71 for definitions.

Table 71 – Acknowledge event notification request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Service Parameter
Event Number	4	Unsigned32	4	Service Parameter

4.3.5.4.42.2 Response APDU parameters

None.

4.3.5.4.42.3 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.5 LAN redundancy services

4.3.5.5.1 LAN redundancy diagnostic message (unconfirmed service Id = 1)

4.3.5.5.1.1 Request APDU parameters

See Table 72 for definitions.

Table 72 – LAN redundancy diagnostic message request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Device Index	0	Unsigned16	2	Service parameter 0 = Index not assigned 1 – 65535 = Device Index
Number of Interfaces	2	Unsigned8	1	Service parameter.
Transmission Interface	3	Unsigned8	1	Identifies the interface from which this APDU was transmitted 1 = Interface A 2 = Interface B
Diagnostic Message Interval	4	Unsigned 32	4	Service parameter.
SMK Physical Device Tag	8	VisibleString	32	Service parameter. Unused octets in this field are set to ASCII space characters.
Reserved	40	Unsigned8	1	Reserved. Set to zero
Duplicate Detection State	41	Unsigned8	1	Service parameter. Encoded as follows: Bits 8-3: Reserved, set to 0. Bit 2: 1 = Duplicate PD Tag Detected 0 = Duplicate PD Tag Not Detected Bit 1: 1 = Duplicate Device Index Detected 0 = Duplicate Device Index Not Detected
Number of Interface	42	Unsigned16	2	The number of Unsigned32 entries in the four Array of Interface Statuses that follow. The value of this

Parameter name	Octet offset	Data type	Octet length	Description
Statuses				field is 1/32nd of the Max Device Index value configured using the SM Set Assignment service.
Array of Interface AtoA Statuses	44	Array of Unsigned32	4* Number of Interface Statuses	Each 32-bit value represents the status of 32 devices. (see note) 1 bit represents each device. Each bit indicates the status of the transmission path from interface A of the reporting device to interface A of this device. The following values are used for each bit: 0 = Diagnostic messages successfully 1 = Diagnostic messages not received
Array of Interface BtoA Statuses	44 + 4* Number of Interface Statuses	Array of Unsigned32	4* Number of Interface Statuses	Each 32-bit value represents the status of 32 devices. (see note) 1 bit represents each device. Each bit indicates the status of the transmission path from interface B of the reporting device to interface A of this device. The following values are used for each bit: 0 = Diagnostic messages successfully 1 = Diagnostic messages not received
Array of Interface AtoB Statuses	44 + 8* Number of Interface Statuses	Array of Unsigned32	4* Number of Interface Statuses	Each 32-bit value represents the status of 32 devices. (see note) 1 bit represents each device. Each bit indicates the status of the transmission path from interface A of the reporting device to interface B of this device. The following values are used for each bit: 0 = Diagnostic messages successfully 1 = Diagnostic messages not received
Array of Interface BtoB Statuses	44 + 12* Number of Interface Statuses	Array of Unsigned32	4* Number of Interface Statuses	Each 32-bit value represents the status of 32 devices. (see note) 1 bit represents each device. Each bit indicates the status of the transmission path from interface B of the reporting device to interface B of this device. The following values are used for each bit: 0 = Diagnostic messages successfully 1 = Diagnostic messages not received
NOTE The bit that identifies a device is located by its bit position in the array. Dividing the device's Device Index by 32 yields the index of the zero-based Unsigned32 array element. The remainder from the division identifies the specific bit in the Unsigned32 array element, counting from the most significant bit (= remainder 1). Therefore, a device with Device Index 33 would be represented by the most significant bit (bit 32) of the second Unsigned32 element of the array, and a device with Device Index 34 would be represented by the next most significant bit (bit 31) of the second Unsigned32 element of the array.				

4.3.5.5.2 LAN redundancy get information (confirmed service Id = 1)

4.3.5.5.2.1 Service overview

This service is always confirmed and, therefore, always uses the Invoke Id in the Message Trailer.

4.3.5.5.2.2 Request APDU parameters

None.

4.3.5.5.2.3 Response APDU parameters

See Table 73 for definitions.

Table 73 – LAN redundancy get information response APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
LAN Redundancy Attributes Version	0	Unsigned32	4	Service Parameter
Max Sequence Number Difference	4	Unsigned8	1	Service Parameter
LAN Redundancy Flags	5	Unsigned8	1	Service Parameter. Encoded as follows; 0 = False, 1 = True Bit 1 Single Mcast APDU Tx I/F Enabled Bit 2 Crossed Cable Detection Enabled Bit 3 Single Mcast APDU Rcpt I/F Enabled Bit 4 Diagnosis using own APDUs Enabled Bit 5 Load Balancing Enabled Bits 6-8 Reserved, set to 0
Reserved	6	Unsigned16	2	Reserved. Set to zero.
Diagnostic Message Interval	8	Unsigned32	4	Service Parameter
AgingTime	12	Unsigned32	4	Service Parameter
Diagnostic Message Interface A Send Address	16	OctetString	16	This parameter is part of the Diagnostic Message Send Addresses service parameter. It defines the destination address for diagnostic messages sent from interface A.
Diagnostic Message Interface A Receive Address	32	OctetString	16	This parameter is part of the Diagnostic Message Receive Addresses service parameter. It defines the destination address for diagnostic messages received on interface A.
Diagnostic Message Interface B Send Address	48	OctetString	16	This parameter is part of the Diagnostic Message Send Addresses service parameter. It defines the destination address for diagnostic messages sent from interface B.
Diagnostic Message Interface B Receive Address	64	OctetString	16	This parameter is part of the Diagnostic Message Receive Addresses service parameter. It defines the destination address for diagnostic messages received on interface B.

4.3.5.5.2.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.5.3 LAN redundancy put information (confirmed service Id = 2)

4.3.5.5.3.1 Service overview

This service is always confirmed and, therefore, always uses the Invoke Id in the Message Trailer.

4.3.5.5.3.2 Request APDU parameters

Same as LAN Redundancy Get Information response parameters. Only those fields following the APDU Version Number are used by the receiving LAN Redundancy Entity.

4.3.5.5.3.3 Response APDU parameters

Same as Request Message parameters. It includes the updated LAN Redundancy Attributes Version.

4.3.5.5.3.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.5.4 LAN redundancy get statistics (confirmed service Id = 3)

4.3.5.5.4.1 Service overview

This service is always confirmed and, therefore, always uses the Invoke Id in the Message Trailer.

4.3.5.5.4.2 Request APDU parameters

None.

4.3.5.5.4.3 Response APDU parameters

See Table 74 for definitions.

Table 74 – LAN redundancy get statistics request APDU parameters

Parameter name	Octet offset	Data type	Octet length	Description
Diagnostic Messages Received On Interface A	0	Unsigned32	4	Service Parameter
Diagnostic Messages Missed On Interface A	4	Unsigned32	4	Service Parameter
Remote Device Diagnostic Message Receive Faults Detected On Interface A	8	Unsigned32	4	Service Parameter
Diagnostic Messages Received On Interface B	12	Unsigned32	4	Service Parameter
Diagnostic Messages Missed On Interface B	16	Unsigned32	4	Service Parameter
Remote Device Diagnostic Message Receive Faults Detected On Interface B	20	Unsigned32	4	Service Parameter
Number of Crossed Cable	24	Unsigned32	4	This field is number of Unsigned32 values in the List of Crossed Cable Status. 0 means that the list is not

Parameter name	Octet offset	Data type	Octet length	Description
Statuses				present. If the list is present, then this field contains 1/32nd of the value of the Max Device Index assigned to the device in the SM Set Assignment Info message.
Array of Crossed Cable Status	28	Array of Unsigned32	4 * Number of Crossed Cable Statuses	Service Parameter. Each bit in this array (see note) represents a device that sends diagnostic messages to the device that is sending this message (the reporting device). Each remote device's bit represents whether the interface it is using to send diagnostic messages, as indicated by the Interface Used for Transmission of this Diagnostic Message field, is the same interface used by the reporting device to receive the messages. The following values are used for each bit: 0 = Cables not crossed or Crossed Cable Detection Enabled is false. 1 = Cables are crossed (Diagnostic message sent from Interface A was received on Interface B, or diagnostic message sent from Interface B was received on Interface A).
NOTE The bit that identifies a device is located by its bit position in the array. Dividing the device's Device Index by 32 yields the index of the zero-based Unsigned32 array element. The remainder from the division identifies the specific bit in the Unsigned32 array element, counting from the most significant bit (= remainder 1). Therefore, a device with Device Index 33 would be represented by the most significant bit (bit 32) of the second Unsigned32 element of the array, and a device with Device Index 34 would be represented by the next most significant bit (bit 31) of the second Unsigned32 element of the array.				

4.3.5.5.4.4 Error APDU parameters

Defined by the Common Error APDU parameters in 8.3.6.1.

4.3.5.6 Structure of object descriptions

4.3.5.6.1 General

Subclause 4.3.5.6 defines the object descriptions conveyed using Type 5 Get OD and Put OD services. VFDs using 32-bit OD indexes are required to use the HSE object description formats defined below. VFDs using 16-bit OD indexes may use either H1 or HSE object description formats.

The FDA Agent may convert Type 9 formatted object descriptions it receives from local VFDs or from Type 9 devices to the corresponding Type 5 formats, but it is not required to do so. In addition, the FDA Agent may, but is not required to, convert HSE formatted object descriptions that it receives from Type 5 PutOD request messages to Type 9 format for delivery to or forwarding to VFDs that use Type 9 formats.

When an FDA Agent that does not support conversion from Type 5 to Type 9 format receives such a request message requiring conversion, it may simply forward the message and rely on the receiving VFD to return the appropriate error message, or it may return a negative response with error class "od" and error code ="type 5 to type 9 format conversion not supported".

The fields contained in each OD construct below are defined in the Type 9 Specification.

Their definitions are the same here, however, their *index*, *data type,* and *length* fields have changed to 32-bits, and their order have changed to align on 32-bit boundaries.

For fields defined to have Boolean values, the value 0 indicates "FALSE" and any non-zero value indicates "TRUE".

To maintain compatibility with Type 9 object descriptions, the first field in an object description conveyed by the FDA Agent is the OD Tag. This OD Tag contains either the tag of a Type 9 object description or a tag reserved for Type 5 object descriptions (hex value 0xFFFF). Following the OD Tag is the object description indicated by the tag. Each Type 5 object description uses one of the formats specified in 4.3.5.6.

Each Type 9 object description contains a Type 5 tag plus a Type 5 object description, as specified by Type 9. The Type 5 tag is the encoded Type 9 ASN.1 tag specified for the GetOD Response and the PutOD Request messages. The Type 9 Object Description is specified as packed data.

The Type 9 tag consists of one or two octets. The first four most significant bits of the first octet (most significant byte) has the value 0. The next four bits is the length of the object description. If this 4-bit length field contains a value less than the hex value 0xF, then the H1 tag is one octet and the 4-bit length field specifies the length of the object description that follows.

Following the Type 5 Type Flag field, all Type 5 Object Descriptions have five additional fields. The format of this common Type 5 Object Description header is shown in Table 75.

Table 75 – Object description header

Field name	Octet offset	Data type	Octet length	Description
OD Tag	0	Unsigned16	2	Set to $FFFF_{16}$ to indicate an Type 5 Object Description
Object Code	2	Integer8	1	Object Code of the Object
All Attributes Flag	3	Unsigned8	1	Indicates long form (=1) or short form (=0) Object Description
Object Description Length	4	Unsigned16	2	Total length in octets of the Object Description
Reserved (set to zero)	6	Unsigned16	2	Set to zero
Index32	8	Unsigned32	4	32-bit Object Index

A null object in Type 5 is used to indicate an unused object index, as it is in Type 9. It is represent in Type 5 as shown in Table 76.

Table 76 – Null object

Field name	Value
OD Tag	FF FF_{16}
Object Code	0
All Attributes Flag	0
Object Description Length	12
Reserved (set to zero)	0
Index32	Index of Null Object

4.3.5.6.2 Structure of the list of object descriptions

See Table 77 for definitions.

Table 77 – Structure of the list of object descriptions

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF_{16}	0	Unsigned16	2
Object Code = 16	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
ROM/RAM Flag	12	Unsigned8	1
Access Protection Supported	13	Unsigned8	1
Reserved (set to zero)	14	Unsigned16	2
Name Length	16	Unsigned32	4
Version OD	20	Unsigned32	4
ST-OD Length	24	Unsigned32	4
First Index S-OD	28	Unsigned32	4
S-OD Length	32	Unsigned32	4
First Index DV-OD	36	Unsigned32	4
DV-OD Length	40	Unsigned32	4
First Index DP-OD	44	Unsigned32	4
DP-OD Length	48	Unsigned32	4
Long Form Additional Fields (see note)	52		
Local Address OD-ODES	0 + Long Form Offset	Unsigned64	8
Local Address ST-OD	8 + Long Form Offset	Unsigned64	8
Local Address S-OD	16 + Long Form Offset	Unsigned64	8
Local Address DV-OD	24 + Long Form Offset	Unsigned64	8
Local Address DP-OD	32 + Long Form Offset	Unsigned64	8
NOTE If the Short Form is requested, these fields are not present.			

4.3.5.6.3 Structure of a load region in the S-OD

See Table 78 for definitions.

Table 78 – Structure of a load region in the S-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF_{16}	0	Unsigned16	2
Object Code = 18	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Domain State	12	Unsigned8	1
Upload State	13	Unsigned8	1
Reserved (set to zero)	14	Unsigned16	2

Field name	Octet offset	Data type	Octet length
Max Octets	16	Unsigned32	4
Counter	20	Unsigned32	4
Long Form Additional Fields (see note 1)	24		
Local Address	0 + Long Form Offset	Unsigned64	8
Password	8 + Long Form Offset	Unsigned8	1
Access Groups	9 + Long Form Offset	Unsigned8	1
Access Rights	10 + Long Form Offset	Unsigned16	2
Extension Length	12 + Long Form Offset	Unsigned32	4
Extension Data	16 + Long Form Offset	OctetString	Extension Length
Domain Name	16 + Long Form Offset + Extension Length	VisibleString	Name Length (see note 2)
NOTE 1 If the Short Form is requested, these fields are not present.			
NOTE 2 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.5.6.4 Structure of a function invocation in the DP-OD

See Table 79 for definitions.

Table 79 – Structure of a function invocation in the DP-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 19	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Deletable	12	Unsigned8	1
Reusable	13	Unsigned8	1
PI-State	14	Unsigned8	1
Reserved	15	Unsigned8	3
Number Of Domains	16	Unsigned32	4
List of Domain Indexes	20	Unsigned32	Number Of Domains x 4 (see note 1)
Long Form Additional Fields (see note 2)	20 + (Number Of Domains x 4)		
Password	0 + Long Form Offset	Unsigned8	1
Access Groups	1 + Long Form Offset	Unsigned8	1
Access Rights	2 + Long Form Offset	Unsigned16	2
Extension Length	4 + Long Form Offset	Unsigned32	4
Extension Data	8 + Long Form Offset	OctetString	Extension Length
PI Name	8 + Long Form Offset + Extension Length	VisibleString	Name Length (see note 3)
NOTE 1 If Number Of Domains is 0, then this field is not present.			
NOTE 2 If the Short Form is requested, these fields are not present.			
NOTE 3 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.5.6.5 Structure of an event in the S-OD

See Table 80 for definitions.

Table 80 – Structure of an event in the S-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 20	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Reserved	12	OctetString	3
Enabled	15	Unsigned8	1
Index Event Data	16	Unsigned32	4
Length	20	Unsigned32	4
Long Form Additional Fields (see note 1)	24		
Password	0 + Long Form Offset	Unsigned8	1
Access Groups	1 + Long Form Offset	Unsigned8	1
Access Rights	2 + Long Form Offset	Unsigned16	2
Extension Length	4 + Long Form Offset	Unsigned32	4
Extension Data	8 + Long Form Offset	OctetString	Extension Length
Event Name	8 + Long Form Offset + Extension Length	VisibleString	Name Length (see note 2)
NOTE 1 If the Short Form is requested, these fields are not present.			
NOTE 2 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.5.6.6 Structure of a data type in the ST-OD

See Table 81 for definitions.

Table 81 – Structure of a data type in the ST-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 21	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Reserved	10	Unsigned16	2
Length of Data Type	12	Unsigned32	4
Symbolic Name Length	16	Unsigned32	4
*Symbolic Name	20	VisibleString	Symbolic Name Length
NOTE If the Symbolic Name Length is 0, then this field is not present.			

4.3.5.6.7 Structure of a data type structure description in the ST-OD

See Table 82 for definitions.

Table 82 – Structure of a data type structure description in the ST-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 22	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Reserved (set to zero)	12	Unsigned16	2
Number of Elements	14	Unsigned16	2
List of Elements	16	Structure	8 x Number of Elements
Data Type Index		Unsigned32	4
Length		Unsigned32	4

4.3.5.6.8 Structure of a simple variable in the S-OD

See Table 83 for definitions.

Table 83 – Structure of a simple variable in the S-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 23	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Data Type Index	12	Unsigned32	4
Length	16	Unsigned32	4
Long Form Additional Fields (see note 1)	20		
Local Address	0 + Long Form Offset	Unsigned64	8
Password	8 + Long Form Offset	Unsigned8	1
Access Groups	9 + Long Form Offset	Unsigned8	1
Access Rights	10 + Long Form Offset	Unsigned16	2
Extension Length	12 + Long Form Offset	Unsigned32	4
Extension Data	16 + Long Form Offset	OctetString	Extension Length
Variable Name	16 + Long Form Offset + Extension Length	VisibleString	Name Length (see note 2)
NOTE 1 If the Short Form is requested, these fields are not present.			
NOTE 2 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.5.6.9 Structure of an array in the S-OD

See Table 84 for definitions.

Table 84 – Structure of an array in the S-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 24	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Number of Elements	12	Unsigned32	2
Data Type Index	16	Unsigned32	4
Length	20	Unsigned32	4
Long Form Additional Fields (see note 1)	24		
Local Address	0 + Long Form Offset	Unsigned64	8
Password	8 + Long Form Offset	Unsigned8	1
Access Groups	9 + Long Form Offset	Unsigned8	1
Access Rights	10 + Long Form Offset	Unsigned16	2
Extension Length	12 + Long Form Offset	Unsigned32	4
Extension Data	16 + Long Form Offset	OctetString	Extension Length
Variable Name	16 + Long Form Offset + Extension Length	VisibleString	Name Length (see note 2)
NOTE 1 If the Short Form is requested, these fields are not present.			
NOTE 2 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.5.6.10 Structure of a record in the S-OD

See Table 85 for definitions.

Table 85 – Structure of a record in the S-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF$_{16}$	0	Unsigned16	2
Object Code = 25	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Data Type Index	12	Unsigned32	4
Reserved	16	Unsigned32	4
Long Form Additional Fields (see note 1)	20		
List of Local Addresses	0 + Long Form Offset	Unsigned64	K x 8

Field name	Octet offset	Data type	Octet length
			(see note 2)
Password	(K x 8) + Long Form Offset	Unsigned8	1
Access Groups	1 + (K x 8) + Long Form Offset	Unsigned8	1
Access Rights	2 + (K x 8) + Long Form Offset	Unsigned16	2
Extension Length	4 + (K x 8) + Long Form Offset	Unsigned32	4
Extension Data	8 + (K x 8) + Long Form Offset	OctetString	Extension Length
Record Name	8 + (K x 8) + Long Form Offset + Extension Length	VisibleString	Name Length (see note 3)
NOTE 1 If the Short Form is requested, these fields are not present.			
NOTE 2 K represents the number of fields in the record.			
NOTE 3 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.5.6.11 Structure of a variable list in the DV-OD

See Table 86 for definitions.

Table 86 – Structure of a variable list in the DV-OD

Field name	Octet offset	Data type	Octet length
Short Form Fields			
OD Tag = FF FF_{16}	0	Unsigned16	2
Object Code = 26	2	Integer8	1
All Attributes Flag	3	Unsigned8	1
Object Description Length	4	Unsigned16	2
Reserved (set to zero)	6	Unsigned16	2
Index32	8	Unsigned32	4
Deletable	12	Unsigned8	1
Reserved	13	Unsigned8	1
Number of Elements	14	Unsigned16	2
List of Element Indexes	16	Unsigned32	4 x Number of Elements
Long Form Additional Fields (see note 1)	20 + (K x 4) (see note 2)		
Password	0 + Long Form Offset	Unsigned8	1
Access Groups	1 + Long Form Offset	Unsigned8	1
Access Rights	2 + Long Form Offset	Unsigned16	2
Extension Length	4 + Long Form Offset	Unsigned32	4
Extension Data	8 + Long Form Offset	OctetString	Extension Length
Variable List Name	8 + Long Form Offset + Extension Length	VisibleString	Name Length (see note 3)
NOTE 1 If the Short Form is requested, these fields are not present.			
NOTE 2 K represents the Number of Elements in the record.			
NOTE 3 If Name Length in the OD Object Description is 0, then this field is not present.			

4.3.6 Error APDU bodies

4.3.6.1 Common error parameters

The following error parameter format as shown in Table 87 is used for many services.

Table 87 – Common error parameters

Parameter name	Octet offset	Data type	Octet length	Description
Error Class	0	Unsigned8	1	This parameter identifies the category of the error. Its values are defined in the Error Class production of the Type 9 Abstract Syntax. The values are the tagged values. The value 9 has been added for the Type 5 Initiate Error APDU use.
Error Code	1	Unsigned8	1	This parameter identifies the specific type of error within the class Its values are defined in the Error Class production of the Type 9 Abstract Syntax. The values are the enumerated values with parenthesized values, such as other = 0. Values for error class 11 have been added for Initiate Error message use.
Additional Code	3	Integer16	1	This parameter provides an additional code that is associated with the error.
Additional Description	4	VisibleString	16	This parameter provides 16 octets of additional information that is associated with the error.

4.3.6.2 PI error parameters

See Table 88 for definitions.

Table 88 – PI error parameters

Parameter name	Octet offset	Data type	Octet length	Description
Pi State	0	Unsigned8	1	Defined in Service Parameter.
Reserved	1	OctetString	3	Reserved, set to 0
Error Class	4	Unsigned8	1	Defined in Service Parameter.
Error Code	5	Unsigned8	1	Defined in Service Parameter.
Additional Code	7	Integer16	1	Defined in Service Parameter.
Additional Description	8	VisibleString	16	Defined in Service Parameter.

4.3.6.3 OD error parameters

See Table 89 for definitions.

Table 89 – OD error parameters

Parameter name	Octet offset	Data type	Octet length	Description
Index	0	Unsigned32	4	Defined in the Service Parameter.
Error Class	4	Unsigned8	1	Defined in the Service Parameter.
Error Code	5	Unsigned8	1	Defined in the Service Parameter.
Additional Code	7	Integer16	1	Defined in the Service Parameter.
Additional Description	8	VisibleString	16	Defined in the Service Parameter.

4.3.6.4 Error class and error code values

Table 90 specifies the FDA Agent error codes for each error class.

Table 90 – Error class and error code values

Description:		Values:	
Class	**Code**	**Class**	**Code**
vfd state	other	1	0
application reference	other	2	0
	application unreachable	2	1
definition	other	3	0
	object undefined	3	1
	object attributes inconsistent	3	2
	name already exists	3	3
resource	other	4	0
	memory unavailable	4	1
	max outstanding requests per session exceeded	4	2
	max sessions exceeded	4	3
	object creation failure	4	4
service	other	5	0
	object state conflict	5	1
	pdu size	5	2
	object constraint conflict	5	3
	parameter inconsistent	5	4
	illegal parameter	5	5
	unsupported service	5	6
	unsupported version	5	7
	invalid options	5	8
	unsupported protocol	5	9
	reserved	5	10
	key parameter mismatch	5	11
	assignments already made	5	12
	unsupported device redundancy state	5	13
	response time-out	5	14
	duplicate PD Tag detected	5	15
access	other	6	0
	object invalidated	6	1
	hardware fault	6	2
	object access denied	6	3
	invalid address	6	4
	object attribute inconsistent	6	5
	object access unsupported	6	6
	object non existent	6	7
	type conflict	6	8

Description:		**Values:**	
Class	**Code**	**Class**	**Code**
	named access unsupported	6	9
	access to element unsupported	6	10
	config access already open	6	11
	reserved	6	12
	unrecognized FDA Address	6	13
od	other	7	0
	name length overflow	7	1
	od overflow	7	2
	od write protected	7	3
	extension length overflow	7	4
	od description length overflow	7	5
	operational problem	7	6
	type 5 to type 9 format conversion not supported	7	7
other	other	8	0
reject	pdu size	9	5
sm reason code	other	10	0
	DLL Error – insufficient resources	10	1
	DLL Error – sending queue full	10	2
	DLL Error – time-out before transmission	10	3
	DLL Error – reason unspecified	10	4
	Device failed to respond to SET_PD_TAG	10	5
	Device failed to respond to WHO_HAS_PD_TAG	10	6
	Device failed to respond to SET_ADDR	10	7
	Device failed to respond to IDENTIFY	10	8
	Device failed to respond to ENABLE_SM_OP	10	9
	Device failed to respond to CLEAR_ADDRESS	10	10
	Multiple Response from WHO_HAS_PD_TAG	10	11
	Non-Matching PD_TAG from WHO_HAS_PD_TAG	10	12
	Non-Matching PD_TAG from IDENTIFY	10	13
	Non-Matching DEV_ID from IDENTIFY	10	14
	Remote Error Invalid State	10	15
	Remote Error PD-Tag doesn’t match	10	16
	Remote Error Dev-ID doesn’t match	10	17
	Remote Error SMIB object write failed	10	18
	Remote Error Starting SM Operational	10	19
initiate	other	11	0
	max-fms-pdu-size-insufficient	11	1
	feature-not-supported	11	2
	version-od-incompatible	11	3
	user-initiate-denied	11	4
	password-error	11	5
	profile-number-incompatible	11	6

4.4 FAL protocol state machine structure

The FAL Protocol state machine structure defined for Type 9 applies.

4.5 SMK state machine

4.5.1 General

The SMK protocol machine (SMKPM) defines the behavior of the SMK for those states that affect its communications.

In the protocol machine that follows:

1. The Invoke Id parameter is not included explicitly. Instead, it is considered to be an integral part of all confirmed service and Find Tag primitives. It is received from the sending SMK user and conveyed implicitly to the remote SMK user.
2. The response address is not included explicitly. It is always taken from the source network address of the request.
3. To send and receive SMK APDUs, ARs are not used. Instead, the SMKPM performs the necessary AR functions.
4. The double forward slash, i.e. "//" marks the beginning of a comment in a line.

4.5.2 Messages received by the SMKPM

The SMKPM receives APDUs from the FDA Agent. These APDUs are modeled as Events of SMKPM transitions. The RcvMsg() function is used to represent the arrival of an APDU.

4.5.3 SMKPM service primitives and local APs, Type 5 SMK service processors, and Type "X" interface functions

Table 91 shows the service primitives (in alphabetic order) used in the SMKPM. They are used as follows.

- Request service primitives are used as Events of SMKPM transitions. They are received from Local APs.
- Request service primitives are used also in Actions of SMKPM transitions. When used in this fashion, they cause the corresponding event to be delivered to the SMKPM and are processed immediately, ahead of all other queued events.
- Indication service primitives are used in Actions of SMKPM transtions. They may be delivered to Local APs or to Type "X" Interface Functions. There is one interface function for each Type "X" interface in a linking device. Each is identified by the link id of the FDA Address contained in the received APDU. Link Id 00 00 when used with the Type "X" NN.SS SMK group address means all Type "X" Interface Functions. Service names preceded by "LD_" identify services provided by Type "X" Interface Functions. These services take the sm_svc parameter of the received APDU as a parameter. They invoke one or more Type "X" SM services to satisfy requests received from the Type 5 SMK. Issuing one of these service requests causes the Type 5 SMK to save the Invoke Id so that it can match the response with the request. The Type "X" services used and their sequence is implementation dependent. If the Type "X" Interface Function receives an LD_FindTagQuery request and locates the queried object, it responds by issuing an SM_FindTagReply request.
- Response service primitives are used as Events of SMKPM transitions. They may be received from a Local AP or from an Interface Functions.

- Response service primitives are used also in Actions of SMKPM transitions. When used in this fashion, they cause the corresponding event to be delivered to the SMKPM and are processed immediately, ahead of all other queued events.
- Confirmation service primitives are used in Actions of SMKPM transtions. They are delivered to the requesting Local APs.

Table 91 – SMKPM service primitives

Service	Primitives			
	.req	.ind	.rsp[a]	.cnf
Type_X_ClearAddress		X	X	
Type_X_ClearAssignmentInfo		X	X	
Type_X_FindTagQuery		X		
Type_X_FindTagReply		X		
Type_X_SetAssignmentInfo		X	X	
SM_ClearAddress	X	X	X	X
SM_ClearAssignmentInfo	X	X	X	X
SM_DeviceAnnunciation	X			
SM_FindTagQuery	X	X		
SM_FindTagReply	X	X		
SM_Identify	X	X	X	X
SM_SetAssignmentInfo	X	X	X	X
[a]. rsp is the positive response primitive as well as the negative response primitive.				

4.5.4 State table transition processing

The state tables that follow are composed of a series of numbered transitions. The order of processing the transitions is based on the following rules.

- When receiving an event, locate the first transition in the table containing an instance of the event. A communications event is identified by its indication, e.g. SM_Clear Address.ind.
- Beginning with that transition, proceed down the state table, transition by transition, until a transition is located in which its Current state matches the current state of the state machine and its Event matches the received event.
- Test the conditions. If the result is false, proceed down to the next transition that matches the Current state and Event.
- If no transition matches, then ignore the event.
- When a transition is located, perform the specified action and change the current state of the state machine to the Next state specified by the transition. Then wait for the next event.

4.5.5 States

Table 92 and Figure 1 define the states and their transitions.

Table 92 – SMKPM states

No Address	The device does not have an network address on any of its Type 5 interfaces. If it has an network address on any of its Type 5 interfaces, it is not in this state. In this state, the SMKPM waits for an event that receives an network address through a local interface from the underlying layers.
No Tag	The device does not have valid assignment info. In this state, the SMK sends Annunciate requests and listens for a valid Set Assignment Info APDU. It also responds to Clear Address and SM Identify Request APDUs.
Operational	The device has an network address and valid assignment info. This is the normal operating state of the SMK.

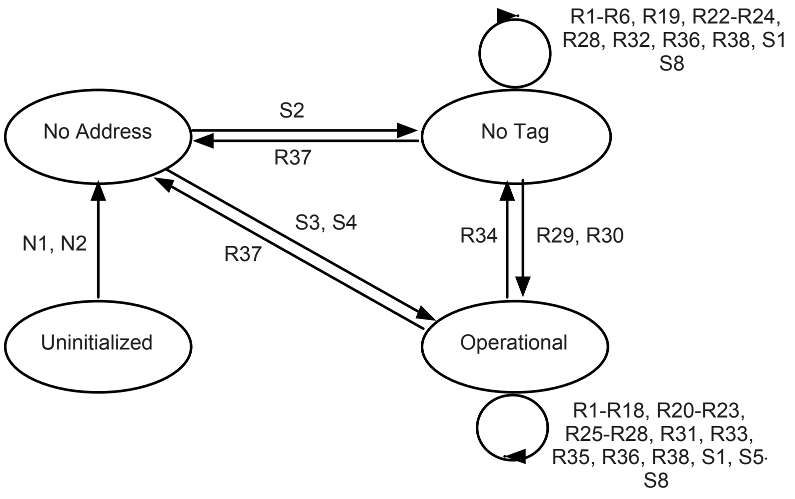
R1-R6, R19, R22-R24, R28, R32, R36, R38, S1 S8
No Address
S2
R37
No Tag
S3, S4
N1, N2
R37
R34
R29, R30
Uninitialized
Operational
R1-R18, R20-R23, R25-R28, R31, R33, R35, R36, R38, S1, S5-S8

Figure 1 – State transition diagram for SMK

4.5.6 State tables

Table 93 through Table 95 define the state machine.

Table 93 – SMKPM state table – initialization

#	Current state	Event or condition => action	Next state
6) N1	7) Uninitialized	8) Powerup 9) && OperationalRestore () = "false" 10) => 11) RestoreDefaults ()	12) No Address
13) N2	14) Uninitialized	15) Powerup 16) && OperationalRestore () = "true" 17) => 18) // No action	19) No Address

Table 94 – SMKPM state table – receive transitions

#	Current state	**Event** or **condition** **=> action**	Next state
20) R 1	21) Operational 22) No Tag	23) RcvMsg() = "Any Unconfirmed SM Req Message" 24) && FdaAddressType (sm_svc) = "UNRECOGNIZED" 25) => 26) // Do nothing – no response for unconfirmed messages	27) Same
28) R 2	29) Operational 30) No Tag	31) RcvMsg() = "Any Confirmed SM Req Message" 32) && FdaAddressType (sm_svc) = "UNRECOGNIZED" 33) => 34) SM_ConfirmedService.err { 35) sm_service_type = SvcType (sm_svc), 36) error_class = "access" 37) error_code = "unrecognized FDA Address" 38) addl_code = 2 39) }	40) Same
41) R 3	42) Operational 43) No Tag	44) RcvMsg() = "Any Confirmed SM Req Message" 45) && FdaAddressType (sm_svc) = "GROUP SMK" 46) => 47) SM_ConfirmedService.err { 48) sm_service_type = SvcType (sm_svc), 49) error_class = "access" 50) error_code = "unrecognized FDA Address" 51) addl_code = 3 52) }	53) Same
54) R 4	55) Operational 56) No Tag	57) RcvMsg() = "Any Confirmed SM Req Message" 58) && IsValid (sm_svc) = "false" 59) => 60) SM_ConfirmedService.err { 61) sm_service_type = SvcType (sm_svc), 62) error_class = "service" 63) error_code = "parameter inconsistent" 64) addl_code = GetAddlCode () 65) }	66) Same
67) R 5	68) Operational 69) No Tag	70) RcvMsg() = "Any Confirmed SM Rsp Message" || 71) RcvMsg() = "Any Confirmed SM Error Message" 72) => 73) SM_ConfirmedService.cnf {} * 74) 75) * How the SMKPM locates the requesting AP is not specified. There may be more than one way to do this. Error checking is performed on the response by the requesting AP.	76) Same
77) R 6	78) Operational 79) No Tag	80) RcvMsg() = "SM_FindTagQuery" 81) && IsValid (sm_svc) = “false” 82) => 83) // Do nothing – no response for unconfirmed messages	84) Same
85) R7	86) Operational	87) RcvMsg() = "SM_FindTagQuery" 88) && (FdaAddressType (sm_svc) = "HSE SMK" 89) || FdaAddressType (sm_svc) = "GROUP SMK") 90) && DeviceRedundancyState () = "secondary" 91) && (QueryType (sm_svc) = “Secondary PD Tag” 92) || QueryType (sm_svc) = “Device Index”) 93) && QueryMatch (sm_svc) = "true" 94) => 95) SM_FindTagReply.req {}	96) Same

97)
R8
	98) Operational	99) RcvMsg() = "SM_FindTagQuery"
		100) && DeviceRedundancyState () = "secondary"
		101) =>
		102) // Discard message and do nothing. Secondaries do not respond to
		103) // any other queries
			104) Same

105) R9	106) Operational	107) RcvMsg() = "SM_FindTagQuery" 108) && (FdaAddressType (sm_svc) = "HSE SMK" 109) || FdaAddressType (sm_svc) = "GROUP SMK") 110) && (QueryType (sm_svc) = "Device Index" 111) || QueryType (sm_svc) = "VFD Reference") 112) && QueryMatch (sm_svc) = "true" 113) => 114) SM_FindTagReply.req {}	115) Same
116) R10	117) Operational	118) RcvMsg() = "SM_FindTagQuery" 119) && FdaAddressType (sm_svc) = "HSE SMK"	125) Same

#	Current state	Event or condition => action	Next state
		120) && (QueryType (sm_svc) = “Primary PD Tag” 121) || QueryType (sm_svc) = “VFD Tag”) 122) && QueryMatch (sm_svc) = "true" 123) => 124) SM_FindTagReply.req {}	
126) R11	127) Operational	128) RcvMsg() = "SM_FindTagQuery" 129) && FdaAddressType (sm_svc) = "GROUP SMK" 130) && (QueryType (sm_svc) = “Primary PD Tag” || 131) QueryType (sm_svc) = “VFD Tag”) 132) && QueryMatch (sm_svc) = "true" 133) => 134) SM_FindTagReply.req {} 135) Type_X_FindTagQuery.ind {}	136) Same
137) R12	138) Operational	139) RcvMsg() = "SM_FindTagQuery" 140) && FdaAddressType (sm_svc) = "GROUP SMK" 141) && (QueryType (sm_svc) = “Primary PD Tag” 142) || QueryType (sm_svc) = “VFD Tag”) 143) && QueryMatch (sm_svc) = "false" 144) => 145) Type_X_FindTagQuery.ind {}	146) Same
147) R13	148) Operational	149) RcvMsg() = "SM_FindTagQuery" 150) && FdaAddressType (sm_svc) = "HSE SMK" 151) && (QueryType (sm_svc) = “Function Block Tag” 152) || QueryType (sm_svc) = "Element ID") 153) => 154) SM_FindTagQuery.ind {}	155) Same
156) R14	157) Operational	158) RcvMsg() = "SM_FindTagQuery" 159) && FdaAddressType (sm_svc) = "GROUP SMK" 160) && (QueryType (sm_svc) = “Function Block Tag” 161) || QueryType (sm_svc) = "Element ID") 162) => 163) SM_FindTagQuery.ind {} 164) Type_X_FindTagQuery.ind {}	165) Same
166) R15	167) Operational	168) RcvMsg() = "SM_FindTagQuery" 169) && (FdaAddressType (sm_svc) = "Type_X SMK" 170) || FdaAddressType (sm_svc) = "LD Type_X INTERFACE SMK" 171) && (QueryType (sm_svc) = “Primary PD Tag” 172) || QueryType (sm_svc) = “Function Block Tag” 173) || QueryType (sm_svc) = "Element ID" 174) || QueryType (sm_svc) = "VFD Tag") 175) => 176) Type_X_FindTagQuery.ind {}	177) Same
178) R16	179) Operational	180) RcvMsg() = "SM_FindTagReply" 181) && FdaAddressType (sm_svc) = "HSE SMK" 182) && DuplicateQueryIdMatch (sm_svc) = "true" 183) && DevId_Match (sm_svc) = "false" 184) => 185) Set_DuplicatePdTagFlag () 186) SM_DeviceAnnunciation.req {} 187) Restart_HseRepeatTimer ()	188) Same
189) R17	190) Operational	191) RcvMsg() = "SM_FindTagReply" 192) && FdaAddressType (sm_svc) = "HSE SMK" 193) && DuplicateQueryIdMatch (sm_svc) = "true" 194) && DevId_Match (sm_svc) = “true” 195) => 196) // Do nothing – the response is from this device	197) Same
198) R18	199) Operational	200) RcvMsg() = "SM_FindTagReply" 201) && FdaAddressType (sm_svc) = "HSE SMK" 202) => 203) SM_FindTagReply.ind {} 204) 205) * Deliver indication to the AP that issued the Find Tag Query. How the SMKPM locates the requesting AP is not specified. There may be more than one way to do this. Error checking is performed on the response by the requesting AP.	206) Same
207) R19	208) No Tag	209) RcvMsg() = "SM_IdentifyReq" 210) && (FdaAddressType (sm_svc) = "Type_X SMK" 211) || FdaAddressType (sm_svc) = "LD Type_X INTERFACE SMK") 212) => 213) SM_ConfirmedService.err { 214) sm_service_type = SvcType (sm_svc),	219) Same

#	Current state	Event or condition => action	Next state
		215) error_class = "service" 216) error_code = "object state conflict " 217) addl_code = 19 218) }	
220) R20	221) Operational	222) RcvMsg() = "SM_IdentifyReq" 223) && FdaAddressType (sm_svc) = "Type_X SMK" 224) && SmCacheEntry(sm_svc) = "false" 225) => 226) SM_ConfirmedService.err { 227) sm_service_type = SvcType (sm_svc), 228) error_class = "access" 229) error_code = "object invalidated" 230) addl_code = 20 231) }	232) Same
233) R21	234) Operational 235) No Tag	236) RcvMsg() = "SM_IdentifyReq" 237) => 238) SM_Identify.rsp {}	239) Same
240) R22	241) Operational 242) No Tag	243) (RcvMsg() = "SM_SetAssignmentInfoReq" 244) || RcvMsg() = "SM_ClearAssignmentInfoReq" 245) || RcvMsg() = "SM_ClearAddressReq") 246) && FdaAddressType (sm_svc) = "LD Type_X INTERFACE SMK" 247) => 248) SM_ConfirmedService.err { 249) sm_service_type = SvcType (sm_svc), 250) error_class = "access" 251) error_code = "object access denied" 252) addl_code = 22 253) }	254) Same
255) R23	256) Operational 257) No Tag	258) (RcvMsg() = "SM_SetAssignmentInfoReq" 259) || RcvMsg() = "SM_ClearAssignmentInfoReq" 260) || RcvMsg() = "SM_ClearAddressReq") 261) && DevId_Match (sm_svc) = “false” 262) => 263) SM_ConfirmedService.err { 264) sm_service_type = SvcType (sm_svc), 265) error_class = "service" 266) error_code = “key parameter mismatch” 267) addl_code = GetAddlCode () 268) }	269) Same
270) R24	271) No Tag	272) (RcvMsg() = "SM_SetAssignmentInfoReq" 273) || RcvMsg() = "SM_ClearAssignmentInfoReq" 274) || RcvMsg() = "SM_ClearAddressReq") 275) && FdaAddressType (sm_svc) = "Type_X SMK" 276) => 277) SM_ConfirmedService.err { 278) sm_service_type = SvcType (sm_svc), 279) error_class = "service" 280) error_code = “object state conflict” 281) addl_code = 24 282) }	283) Same
284) R25	285) Operational	286) (RcvMsg() = "SM_SetAssignmentInfoReq" 287) || RcvMsg() = "SM_ClearAssignmentInfoReq" 288) || RcvMsg() = "SM_ClearAddressReq") 289) && FdaAddressType (sm_svc) = "Type_X SMK" 290) && DeviceRedundancyState () = "secondary" 291) => 292) SM_ConfirmedService.err { 293) sm_service_type = SvcType (sm_svc), 294) error_class = "access" 295) error_code = “object access denied” 296) addl_code = 25 297) }	298) Same
299) R26	300) Operational	301) (RcvMsg() = "SM_SetAssignmentInfoReq" 302) || RcvMsg() = "SM_ClearAssignmentInfoReq" 303) || RcvMsg() = "SM_ClearAddressReq") 304) && PdTag_Match() = "false" 305) => 306) SM_ConfirmedService.err { 307) sm_service_type = SvcType (sm_svc), 308) error_class = "service" 309) error_code = “key parameter mismatch” 310) addl_code = GetAddlCode ()	312) Same

#	Current state	Event or condition => action	Next state
		311) }	
313) R27	314) Operational	315) RcvMsg() = "SM_SetAssignmentInfoReq" 316) && FdaAddressType (sm_svc) = "Type_X SMK" 317) => 318) Type_X_SetAssignmentInfo.ind{}	319) Same
320) R28	321) Operational 322) No Tag	323) RcvMsg() = "SM_SetAssignmentInfoReq" 324) && PdTagDeviceIndex_Check(sm_svc) = "false" 325) => 326) SM_ConfirmedService.err { 327) sm_service_type = SvcType (sm_svc), 328) error_class = "service" 329) error_code = “parameter inconsistent” 330) addl_code = GetAddlCode () 331) }	332) Same
333) R29	334) No Tag	335) RcvMsg() = "SM_SetAssignmentInfoReq" 336) && FdaAddressType (sm_svc) = "HSE SMK" 337) && DeviceRedundancyState () = “Primary” 338) => 339) Set_Assignment_Data (sm_svc) 340) Clear_DuplicatePdTagFlag () 341) SM_SetAssignmentInfo.rsp {} 342) SM_DeviceAnnunciation.req {} 343) Restart_HseRepeatTimer () 344) SM_FindTagQuery.req {}	345) Opera tional
346) R30	347) No Tag	348) RcvMsg() = "SM_SetAssignmentInfoReq" 349) && FdaAddressType (sm_svc) = "HSE SMK" 350) && DeviceRedundancyState () = “Secondary” 351) => 352) Set_Assignment_Data (sm_svc) 353) Clear_DuplicatePdTagFlag () 354) SM_SetAssignmentInfo.rsp {} 355) SM_DeviceAnnunciation.req {} 356) Restart_HseRepeatTimer ()	357) Opera tional
358) R31	359) Operational	360) RcvMsg() = "SM_SetAssignmentInfoReq" 361) && FdaAddressType (sm_svc) = "HSE SMK" 362) => 363) Set_Assignment_Data (sm_svc) 364) SM_SetAssignmentInfo.rsp {} 365) SM_DeviceAnnunciation.req {} 366) Restart_HseRepeatTimer ()	367) Same
368) R32	369) No Tag	370) RcvMsg() = "SM_ClearAssignmentInfoReq" 371) && FdaAddressType (sm_svc) = "HSE SMK" 372) => 373) // This will force the device back to defaults 374) RestoreDefaults () 375) SM_ClearAssignmentInfo.rsp {} 376) SM_DeviceAnnunciation.req {} 377) Restart_HseRepeatTimer ()	378) Same
379) R33	380) Operational	381) RcvMsg() = "SM_ClearAssignmentInfoReq" 382) && FdaAddressType (sm_svc) = "Type_X SMK" 383) => 384) Type_X_ClearAssignmentInfo.ind{}	385) Same
386) R34	387) Operational	388) RcvMsg() = "SM_ClearAssignmentInfoReq" 389) && FdaAddressType (sm_svc) = "HSE SMK" 390) => 391) RestoreDefaults () 392) SM_ClearAssignmentInfo.rsp {} 393) SM_DeviceAnnunciation.req {} 394) Restart_HseRepeatTimer ()	395) No Tag
396) R35	397) Operational	398) RcvMsg() = "SM_ClearAddressReq" 399) && FdaAddressType (sm_svc) = "Type_X SMK" 400) => 401) Type_X_ClearAddress.ind{}	402) Same
403) R36	404) Operational 405) No Tag	406) RcvMsg() = "SM_ClearAddressReq" 407) && FdaAddressType (sm_svc) = "HSE SMK" 408) && (AddressToClear(sm_svc) = "None" 409) || ConfigurationSessionActive = “True”) 410) => 411) SM_ConfirmedService.err { 412) sm_service_type = SvcType (sm_svc), 413) error_class = "service" 414) error_code = “object constraint	417) Same

#	Current state	Event or condition => action	Next state
		conflict" 415) addl_code = GetAddlCode () 416) }	
418) R37	419) Operational 420) No Tag	421) RcvMsg() = "SM_ClearAddressReq" 422) && FdaAddressType (sm_svc) = "HSE SMK" 423) && (NumberOfAssignedAddresses() = 1 424) || AddressToClear(sm_svc) = "All") 425) => 426) SM_ClearAddress.rsp {} 427) Clear_Address (interface_to_clear)	428) No Address
429) R38	430) Operational 431) No Tag	432) RcvMsg() = "SM_ClearAddressReq" 433) && FdaAddressType (sm_svc) = "HSE SMK" 434) && NumberOfAssignedAddresses() > 1 435) => 436) Clear_Address (interface_to_clear) 437) SM_ClearAddress.rsp {} 438) SM_DeviceAnnunciation.req {} 439) Restart_HseRepeatTimer ()	440) Same

Table 95 – SMKPM state table – internal events

#	Current state	Event or condition => action	Next state
441) S1	442) Operational 443) No Tag	444) HseRepeatTimerExpires() 445) => 446) SM_DeviceAnnunciation.req {} 447) Restart_HseRepeatTimer ()	448) Same
449) S2	450) No Address	451) RcvNewNetworkAddress (interface, address) 452) && AssignmentInfo_Set () = "false" 453) => 454) RestoreDefaults () 455) NewAddress (interface, address) 456) SM_DeviceAnnunciation.req {} 457) Restart_HseRepeatTimer ()	458) No Tag
459) S3	460) No Address	461) RcvNewNetworkAddress (interface, address) 462) && AssignmentInfo_Set () = "true" 463) && DeviceRedundancyState () = “Primary” 464) => 465) Clear_DuplicatePdTagFlag () 466) NewAddress (interface, address) 467) SM_DeviceAnnunciation.req {} 468) Restart_HseRepeatTimer () 469) SM_FindTagQuery.req {}	470) Operational
471) S4	472) No Address	473) RcvNewNetworkAddress (interface, address) 474) && AssignmentInfo_Set () = "true" 475) && DeviceRedundancyState () = “Secondary” 476) => 477) Clear_DuplicatePdTagFlag () 478) NewAddress (interface, address) 479) SM_DeviceAnnunciation.req {} 480) Restart_HseRepeatTimer ()	481) Operational

482)
S5
	483) Operational	484) // for new address for second HSE interface or for changing the address of either
		485) // HSE interface
		486) RcvNewNetworkAddress (interface, address)
		487) && NetworkAddressChange (interface, address) = “false”
		488) =>
		489) NewAddress (interface, address)
		490) SM_DeviceAnnunciation.req {}
		491) Restart_HseRepeatTimer ()
			492) Same

493) S6	494) Operational	495) SM_UnconfirmedService.req {} 496) || SM_ConfirmedService.req {} 497) || SM_ConfirmedService.rsp {} 498) || Type_X_ConfirmedService.rsp {} 499) => 500) Send_SM_ReqRspMessage (sm_svc)	501) Same
502) S7	503) Operational	504) SM_ConfirmedService.err{} 505) || Type_X_ConfirmedService.err {} 506) => 507) Send_SM_CommonErrorRsp (508) sm_service_type = SvcType (sm_svc), 509) error_class = "result error class" 510) error_code = “result error code” 511) addl_code = “result error addl_code” 512) addl_description = “result error addl_description”) 513))	514) Same
515) S8	516) Operational 517) No Tag	518) SntpSyncLost() 519) => 520) SM_DeviceAnnunciation.req {} 521) Restart_HseRepeatTimer ()	522) Same

4.5.7 Function descriptions

4.5.7.1 General

The following descriptions define the functions referenced by the preceding state tables.

4.5.7.2 Event functions

4.5.7.2.1 Summary

These functions are *callback* functions used to receive an event. They do not cause any modification of the data received. The other events are Service Primitives defined previously. See Table 96 through Table 99 for definitions of the functions.

4.5.7.2.2 HseRepeatTimerExpires ()

Table 96 – HseRepeatTimerExpires ()

Function	
This function is invoked when the Hse Repeat Timer expires.	
Input parameters	Output parameters
none	none

4.5.7.2.3 RcvNewNetworkAddress (interface, address)

Table 97 – RcvNewNetworkAddress (interface, address)

Function	
This function is invoked when an network address is received from the underlying layers. The input parameters identify the Type 5 interface and the received network address.	
Input parameters	Output parameters
interface, address	none

4.5.7.2.4 RcvMsg ()

Table 98 – RcvMsg ()

Function	
This function is invoked when an APDU is received. It conveys the sm_svc parameter to the state machine. This parameter contains all the fields of the APDU, including the header and trailer fields. The syntax RcvMsg() = "SM_XXX" is used to indicate that an APDU for SM service XXX has been received.	
Input parameters	Output parameters
sm_svc	none

4.5.7.2.5 SntpSyncLost ()

Table 99 – SntpSyncLost ()

Function	
This function is invoked when the state of synchronization of time between the device and the selected remote time server changes from true to false.	
Input parameters	Output parameters
none	none

4.5.7.3 Condition functions

4.5.7.3.1 Summary

These functions are used to test an event. They do not cause any modification of the data tested. See Table 100 through Table 116 for definitions of the functions.

4.5.7.3.2 AddressToClear (sm_svc)

Table 100 – AddressToClear (sm_svc)

Function	
Returns which Type 5 interfaces are being requested to be cleared. The values are OperationalNetworkAddress, Non-OperationalNetworkAddress, All, and None. None indicates that the specified Type 5 interface does not have an address or is not capable of dynamically requesting a new address.	
Input parameters	Output parameters
none	HseInterface

4.5.7.3.3 AssignmentInfo_Set ()

Table 101 – AssignmentInfo_Set ()

Function	
Returns true if the SMK assignment information has been previously set by the Set Assignment service and is currently still valid as defined by that service.	
Input parameters	Output parameters
none	true or false

4.5.7.3.4 ConfigurationSessionActive ()

Table 102 – ConfigurationSessionActive ()

Function	
Returns true if there is a configuration AR established.	
Input parameters	Output parameters
none	true or false

4.5.7.3.5 DeviceRedundancyState ()

Table 103 – DeviceRedundancyState ()

Function	
Returns the redundancy state of the device.	
Input parameters	Output parameters
none	primary or secondary

4.5.7.3.6 DevId_Match (sm_svc)

Table 104 – DevId_Match (sm_svc)

Function	
Returns true if the Device ID in the APDU exactly matches the Device ID of this device.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.7 DuplicateQueryIdMatch (sm_svc)

Table 105 – DuplicateQueryIdMatch (sm_svc)

Function	
Returns true if the Invoke Id matches the Invoke Id of the Find Tag Request that was sent to determine if another device has the same PD Tag as this device.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.8 DuplicatePdTagDetected ()

Table 106 – DuplicatePdTagDetected ()

Function	
Returns true if the device has detected another device with the same Physical Device Tag as this device.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.9 FdaAddressType (sm_svc)

Table 107 – FdaAddressType (sm_svc)

Function	
Evaluates the FDA Address component of the sm_svc parameter and returns which SMK it identifies. For the return values defined below, LL LL identifies an Type "X" link connected to the linking device, and NN is the node address for that link interface. Returns: Type 5 SMK if FDA Address = 00 00.00.SM, where SM is the SMK selector GROUP SMK if FDA Address = 00 00.GG GG or LL LL.GG GG, where GG GG is the Type "X" SMK group address LD Type "X" Interface SMK if FDA Address = LL LL.NN.SM and NN is the node address of an Type "X" interface of the LD, Type "X" SMK if FDA Address = LL LL.NN.SM UNRECOGNIZED if FDA Address is not one of the above	
Input parameters	Output parameters
sm_svc	SMK Identifier

4.5.7.3.10 IsValid (sm_svc)

Table 108 – IsValid (sm_svc)

Function	
Returns true if the parameters in the APDU are valid for the service type. Data is considered valid if it has the correct data type and it conforms to the values defined for it. Parameter values are defined in the SMK service specifications in IEC 61158-5 and in the APDU specifications in this specification. The values are checked for validity in this function. Note the contrast with other condition functions defined for the SMKPM, such as PdTag_Match(sm_svc) that check for the use of the value.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.11 NetworkAddressChange (interface, address)

Table 109 – NetworkAddressChange (interface, address)

Function	
Returns true if the specified interface already has a valid network address.	
Input parameters	Output parameters
interface, address	true or false

4.5.7.3.12 NumberOfAssignedAddresses ()

Table 110 – NumberOfAssignedAddresses ()

Function	
Returns the number of Type 5 Interfaces that currently have an network address.	
Input parameters	Output parameters
none	NumberOfInterfaces

4.5.7.3.13 OperationalRestore ()

Table 111 – OperationalRestore ()

Function	
Returns true if Operational Powerup is true and the last state of the SMKPM was Operational.	
Input parameters	Output parameters
none	true or false

4.5.7.3.14 PdTag_Match (sm_svc)

Table 112 – PdTag_Match (sm_svc)

Function	
Returns true if the PD Tag in the APDU exactly matches the PD Tag of the device identified by the FDA Address of the APDU. Group FDA Addresses cause false to be returned.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.15 PdTagDeviceIndex_Check (sm_svc)

Table 113 – PdTagDeviceIndex_Check (sm_svc)

Function	
Returns false if: 1. the service would cause PD Tag or the Device Index to be cleared or remain clear (both shall be set), or. 2. the PD Tag in the device is set before the service begins and the service is attempting to change it (the device index can be changed, but the PD Tag cannot). Otherwise, returns true.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.16 Query_Match (sm_svc)

Table 114 – Query_Match (sm_svc)

Function	
Returns true if the device contains the queried object.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.3.17 QueryType (sm_svc)

Table 115 – QueryType (sm_svc)

Function	
Returns the Query Type for a Find Tag Query or Find Tag Reply.	
Input parameters	Output parameters
sm_svc	query type

4.5.7.3.18 SmCacheEntry (sm_svc)

Table 116 – SmCacheEntry (sm_svc)

Function	
Returns true if the SMK has cached the identification information for the Type "X" device identified by the fda_address component of the sm_svc parameter.	
Input parameters	Output parameters
sm_svc	true or false

4.5.7.4 Action functions

4.5.7.4.1 Summary

These functions are used in the Action portion of the transitions. See Table 117 through Table 129 for definitions of the functions.

4.5.7.4.2 Clear_Address (interface_to_clear)

Table 117 – Clear_Address (interface_to_clear)

Function	
Close all FDA sessions and Type 9 VCRs for the address to be cleared. Stop the Type 5 Repeat Timer if it is running. Clear the address for the specified Type 5 interface in the local network address array . Request the underlying layers to clear the address. If the Operational network address is no longer valid, set it to the address of the other Type 5 interface if that address is valid.	
Input parameters	Output parameters
interface_to_clear	none

4.5.7.4.3 Clear_DuplicatePdTagFlag ()

Table 118 – Clear_DuplicatePdTagFlag ()

Function	
Performs the equivalent of DuplicateDetectedState := DuplicateDetectedState & NOT “DuplicatePdTag”	
Input parameters	Output parameters
none	none

4.5.7.4.4 Get_AddlCode ()

Table 119 – Get_AddlCode ()

Function	
Determines the value of the additional code variable. These values are defined in 4.5.6.	
Input parameters	Output parameters
none	addl_code

4.5.7.4.5 New_Address (interface, address)

Table 120 – New_Address (interface, address)

Function	
If the existing network address for the specified Type 5 interface is valid, close all client/server FDA sessions and Type 9 VCRs for that address. Set or change the address for the specified Type 5 interface in the local network address array. If the Operational network address is not valid or no longer valid, set it to this address. If the Operational Network Address was valid and was changed, then immediately move all existing publisher and report source VCRs and FDA Sessions to the new Operational Network Address.	
Input parameters	Output parameters
interface, address	none

4.5.7.4.6 Restart_HseRepeatTimer ()

Table 121 – Restart_HseRepeatTimer ()

Function	
This function stops the HseRepeatTimer if it is running, and restarts it with the period of time defined by the Type 5 Repeat Time attribute. When this timer expires, it generates the HseRepeatTimerExpires event for the SM state machine.	
Input parameters	Output parameters
None	None

4.5.7.4.7 Restore_Defaults ()

Table 122 – Restore_Defaults ()

Function	
Restore this device to its ‘as manufactured’ condition, except that network addresses and the Operational network address are retained. This closes all sessions and VCRs and removes all configured and operational data.	
Input parameters	Output parameters
none	none

4.5.7.4.8 Send_SM_CommonErrorRsp (sm_service_type, svc_spec_params)

Table 123 – Send_SM_CommonErrorRsp (sm_service_type, svc_spec_params)

Function	
Builds and sends the error response APDU for the service identified in the sm_service_type parameter.	
Input parameters	Output parameters
sm_service_type, svc_spec_params	none

4.5.7.4.9 Send_SM_ReqRspMessage (sm_svc)

Table 124 – Send_SM_ReqRspMessage (sm_svc)

Function	
Builds and sends the request or response APDU for the service identified in the sm_svc parameter.	
Input parameters	Output parameters
sm_svc	none

4.5.7.4.10 Set_Assignment_Data (sm_svc)

Table 125 – Set_Assignment_Data (sm_svc)

Function	
Copies assignment data for an Type 5 device from the APDU to matching SMK attributes. If the Operational IP Address changes, then immediately move all existing publisher and report source VCRs and FDA Sessions to the Operational IP Address.	
Input parameters	Output parameters
sm_svc	none

4.5.7.4.11 Set_DuplicatePdTagFlag ()

Table 126 – Set_DuplicatePdTagFlag ()

Function	
Sets the DuplicateDetectedState attribute to reflect that this device has detected that this device and another device have the same PD Tag.	
Input parameters	Output parameters
none	none

4.5.7.4.12 SvcType (sm_svc)

Table 127 – SvcType (sm_svc)

Function	
This function decodes the data that is conveyed in the sm_svc parameter and returns the service type.	
Input parameters	Output parameters
sm_svc	service type

4.5.8 Error classes and codes

The Error Classes and Codes defined in 4.3.6.4 apply.

4.5.9 Additional code definition

Table 697 and Table 698 define the values that may be used in the Additional Code octet of the FDA Common Error classes. This definition is local to SM services.

Table 128 – Additional code used by error class and code

FDA error class	FDA error code	Additional code value for SMKPM	Provided By
service	key parameter mismatch	see Table 129	GetAddlCode ()
	parameter inconsistent	see Table 129	GetAddlCode ()
	object constraint conflict	see Table 129	GetAddlCode ()

Table 129 – Additional code parameter IDs

Parameter	Additional code
Clear Duplicate Detection State	5
Device ID	10
Device Index	20
Device Redundancy State	25
Element ID	30
FDA Address	35
Function Block Tag	40
Type "X" New Address	45
Interface to Clear	50
Invoke ID	55
PD Tag	60
Type 5 Repeat Time	65

Parameter	Additional code
LAN Redundancy Addr	70
Max Device Index	75
Operational IP Address	80
Query Type	85
VFD Reference	90
VFD Tag	95
Interface does not have an address	101
Interface has a fixed address	102
Interface has a configuration session	103

4.6 VCR state machine

The Type 5 VCR state machine is defined by the Type 9 VCR State Machine, with the following modifications.

- The Type 5 Initiate service replaces the Type 9 Initiate service. Type 9 Initiate service parameters that are not defined for the Type 5 Initiate service are ignored in the state machine.
- References to the AR Abort service by the Type 9 Connection Establishment state transitions are modified as follows. The Abort request primitive is replaced by the sending of an Type 5 VCR Abort request APDU, and the receipt of a Type 9 Abort indication primitive is replaced by the receipt of an Type 5 VCR Abort request APDU. The only exception to this rule is when an Type 9 Abort.request primitive is issued in response to the receipt of an Type 9 Initiate request PDU when the VCR is in the Connection Established state. In this case, the FDA Agent returns an Type 5 Initiate Error APDU with the same error class and error code supplied in the Type 9 Abort.request primitive.
- The Type 5 VCR Client and Server endpoints use Type 5 Idle APDUs as "keep alive" APDUs as follows.
- Client Type 5 VCR endpoints send Idle request APDUs only when the Idle timer expires.
- Server Type 5 VCR endpoints immediately send Idle Response APDUs when they receive Idle request APDUs.
- Client endpoints restart their Idle Timers whenever they send a Type 5 APDU.
- Server endpoints do not maintain an Idle timer.
- Upon expiration of their Inactivity timer, client and server Type 5 VCR endpoints terminate after sending an Abort APDU.
- Requests to establish an Type 5 VCR of type "Type 9 CLIENT" using the VCR Selector option are rejected if the remote selector address of the Type 9 Client VCR indicates that the remote VFD is the MIB VFD. The error class used is "access" and the error code is "object access denied".
- Server endpoints established for MIB access respond negatively to received Type 9 update service requests using error class "access" and error code "object access denied" if the underlying AR is not a configuration AR. The Type 9 update service requests are:
  – Generic Initiate Download Sequence
  – Generic Download Segment
  – Generic Terminate Download Sequence
  – Initiate Download Sequence
  – Download Segment
  – Terminate Download

– Request Domain Download
– Write
– Write with Subindex

4.7 FAL service protocol machine (FSPM)

4.7.1 Type 5 FSPM

The formal interface between an Type 5 VCR and its underlying AR maintains compatibility with the Type 9 VCR state transitions used for the Type 5 VCR State Machine.

The Type 5 FSPM is defined by the Type 9 FSPM. It is used in its entirety with the following modifications:

- The Confirmed Send and Unconfirmed Send service primitives for sending and receiving have been extended to contain the Type 5_VCR_ID parameter to identify the VCR that uses the session endpoint.
- All references to remote DLCEP addresses are replaced by references to remote addresses. When used with ARs that operate over connections, the remote address parameter identifies the connection.
- The Abort service is not conveyed between the FSPM and the Type 5 VCR state machines. Instead, the Type 5 VCR Abort APDU is conveyed between the two state machines using the Unconfirmed Send service.
- Some service parameter differences exist in the service primitives exchanged between the Type 5 FSPM and the Type 5 ARPM. The service primitives and their parameters are defined in 4.8.1.1.1.

4.8 Application relationship protocol machines (ARPMs)

4.8.1 Client / server session ARPM

4.8.1.1 Primitive definitions

4.8.1.1.1 Primitives exchanged between ARPM and FSPM

Table 130 and Table 131 define the primitives exchanged between the ARPM and the FSPM.

Table 130 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
EST_req	FSPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	arep_id, user_data,	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	arep_id, user_data, service_type	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	vcr_id, abort_detail, abort_identifier, reason_code	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	vcr_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	vcr_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
523) UCS_req	524) FS PM	525) vcr_id, 526) service_type, 527) user_data	528) This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 131 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish confirmation(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish confirmation(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	vcr_id, abort_detail, abort_identifier, reason_code	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	vcr_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf(+)	ARPM	vcr_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation(+) primitive from the ARPM to the FSPM.
CS_cnf(-)	ARPM	vcr_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation(-) primitive from the ARPM to the FSPM.
529) UCS_ind	530) A RPM	531) vcr_id, 532) service_type, 533) user_data	534) This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.

4.8.1.1.2 Parameters of FSPM/ARPM primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 132.

Table 132 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
abort_identifier	This parameter conveys the value that is used for the Abort_Identifier parameter.
reason_code	This parameter conveys the value that is used for the Reason_Code parameter.
abort_detail	This parameter conveys the value that is used for the Abort_Detail parameter.
vcr_id	This parameter is used to unambiguously identify an instance of the VCR that issued a primitive or that is the primitive destination. A means for such identification is not specified by this standard.
535) service_type	536) This parameter conveys the protocol Id and the service within the protocol.

4.8.1.2 DLL mapping of Client / Server AREP Class

4.8.1.2.1 Formal Definition

Subclause 4.8.1.2 describes the mapping of the Client / Server AREP Class to the Socket model. The DLL mapping attributes and their permitted values used with the Client / Server AREP class are defined in 4.8.1.2.

CLASS: **ClientServer**
PARENT CLASS: **AR Endpoint**
ATTRIBUTES:

1	(m)	KeyAttribute:	LocalAndRemoteAddresses
1.1	(m)	KeyAttribute:	LocalAddress
1.2	(m)	KeyAttribute:	RemoteAddress
2	(m)	Attribute:	TransferType (CONN, CNLS)
3	(m)	Attribute	BufferSize
4	(m)	Attribute:	TransmitDelayTime
5	(m)	Attribute:	InactivityCloseTime
6	(m)	Attribute:	HdrOptions
7	(m)	Attribute:	GuardBand
8	(m)	Attribute:	MibConfigurationUse
9	(m)	Attribute:	MaxApduLength
10	(m)	Attribute:	MaxOutstanding
11	(m)	Attribute:	ServerPhysicalDeviceTag

4.8.1.2.2 Attributes

LocalAndRemoteAddresses

This key attribute identifies the AREP. It is composed of the local and remote addresses of the underlying layer for the AREP.

TransferType

This attribute specifies whether the connection-oriented (CONN) or connectionless (CNLS) services of the underlying layer are used by the AREP.

BufferSize

This attribute specifies the size in octets of the session endpoint's send buffer. The buffer holds an integral number of APDUs. If an APDU is received for transfer that would overflow the buffer, the buffer is sent first, and the APDU is then added as the first APDU in the buffer.

TransmitDelayTime

This attribute specifies the amount of time that a sending session endpoint accumulates APDUs to send in its buffer. The time period starts when the first APDU is placed into the transmit buffer after either the buffer has been initialized or after it has been emptied by the previous transmission.

If the endpoint has no APDUs to send (the buffer is empty) and it receives an APDU to send, it loads the APDU into the buffer and waits for additional APDUs to concatenate into the buffer. If the buffer fills before the Transmit Delay Time has been reached, the Type 5 FAL AE stops the timer and sends the buffer. When this time interval has expired and the buffer has not yet filled, it sends the contents of the buffer.

InactivityCloseTime

This attribute is used by Client and Server endpoints to determine when to close their endpoints because of inactivity.

HdrOptions

This attribute specifies which APDU Trailer fields are present in the APDUs sent and received on the session.

GuardBand

This attribute specifies the lower and upper bound for the field being protected from rollover.

MibConfigurationUse

This Boolean attribute specifies, when TRUE, that the AR is used for MIB configuration. This is not a configurable attribute. Client applications are expected to know that they need configuration ARs and request them with the Establish request.

MaxApduLength

This attribute specifies the maximum permitted length in octets of APDUs sent by the CLIENT AREPs and received by SERVER AREPs.

MaxOutstanding

This attribute specifies the maximum number of requests that the receiving endpoint may have outstanding at any point in time. Outstanding means that the endpoint has received a request message, but has not returned the corresponding response message. Its initial value is set by the device manufacturer. If a request message is received that causes this attribute to be exceeded, a negative response is returned with error class = "resource" and error code = "max outstanding requests per session exceeded".

ServerPhysicalDeviceTag

This attribute specifies the SMK PhysicalDeviceTag of the server endpoint.

4.8.1.3 Client / Server AREP state machine

4.8.1.3.1 Client / Server ARPM states

The defined states and their descriptions of the Client / Server ARPM are shown in Table 133 and Figure 2. If the transfer type = CONN, then the AR operates over an underlying connection, and the state machine does not take effect until the underlying connection has been established. If the first message received on the underlying connection is not an Establish request, then the underlying connection is terminated.

Table 133 – Client / Server ARPM states

States	Description
CLOSED	Client and Server AREPs shall be created prior to use. They are created in the CLOSED state. When the FDA Agent receives an OpenSession Request FDA-PDU it dynamically creates a new server AREP and delivers the received FDA-PDU to it. AREPs in the CLOSED state are not capable of sending or receiving FDA-PDUs except for OpenSession FDA-PDUs. Transitioning an AREP to the CLOSED state causes its timers to be stopped, its underlying socket connection to be closed if necessary, and it is then deleted.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
REQUESTING	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING	The AREP has received an Establish Request FAL-PDU, delivered an Establish.ind primitive and is waiting for a response from its user.
537) SAME	538) Indicates that the next state is the same as the current state.

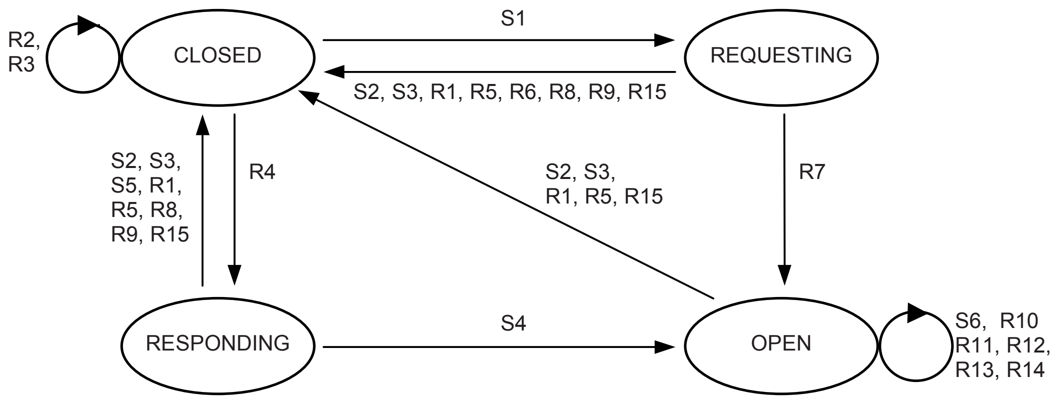

Figure 2 – State transition diagram of client / server ARPM

4.8.1.3.2 Client / server ARPM state table

Table 134 and Table 135 define the Client/Sever ARPM state machine.

Table 134 – Client / server ARPM state table – sender transitions

#	Current state	**Event** or **condition** **=> action**	Next state
539) S1	540) CLOSED	541) EST_req 542) && EndpointType = "CLIENT" 543) => 544) FAL-PDU_req { 545) smpm_service_name = “SMPM_Send”, 546) arep_id = GetArepId (), remote_address = RemoteAddress, 547) fda_pdu = BuildFAL-ReqRspPDU (service_type = “AR Establish Req”, 548) fal_data = user_data) 549) }, 550) 551) StartInactivityCloseTimer()	552) REQUES TING
553) S2	554) RESPONDING 555) REQUESTING 556) OPEN	557) Abort_req 558) =>	559) CLOSED
560) S3	561) REQUESTING RESPONDING 562) OPEN	563) InactivityCloseTimerExpires 564) =>	565) CLOSED
566) S4	567) RESPONDING	568) EST_rsp(+) 569) => 570) FAL-PDU_req { 571) smpm_service_name = “SMPM_Send”, 572) arep_id = GetArepId (), remote_address = RemoteAddress, 573) fda_pdu = BuildFAL-ReqRspPDU (service_type = “AR Establish Rsp”, 574) fal_data = user_data) 575) } 576) 577) StartInactivityCloseTimer()	578) OPEN
579) S5	580) RESPONDING	581) EST_rsp(-) 582) => 583) FAL-PDU_req { 584) smpm_service_name = “SMPM_Send”, 585) arep_id = GetArepId (), remote_address = RemoteAddress, 586) fda_pdu = BuildFAL-ReqRspPDU (service_type = “AR Establish Err”, 587) fal_data = user_data) 588) }	589) CLOSED
590) S6	591) OPEN	592) UCS_req 593) || CS_req 594) || CS_rsp 595) => 596) FAL-PDU_req { 597) smpm_service_name = “SMPM_Send”, 598) arep_id = GetArepId (), remote_address = RemoteAddress, 599) fda_pdu = BuildFAL-ReqRspPDU (service_type = service_type, 600) fal_data = user_data) 601) }	602) OPEN

Table 135 – Client / server ARPM state table – receiver transitions

#	Current state	Event or condition => action	Next state
603) R1	604) REQUESTING 605) RESPONDING OPEN	606) FAL-PDU_ind 607) && FDA_PDU_Valid(fal_pdu) = False 608) => 609) FAL-PDU_req* { 610) smpm_service_name = “SMPM_Send”, 611) arep_id = GetArepId (), 612) remote_address = FAL_Pdu_RemoteAddress(fal_pdu), 613) fal_pdu = BuildFAL-ErrPDU(request_pdu = fal_pdu, 614) error_class = “service”, 615) error_code = appropriate error code for detected error) 616) } 617) 618) * Used only when the received FAL PDU is a DTC Request	619) CLOSED
620) R2	621) CLOSED	622) FAL-PDU_ind 623) && EndpointType = “SERVER” 624) && FAL_Pdu_SvcType (fda_pdu) = “AR Establish Req” 625) && FAL_PDU_Valid(fda_pdu) = False 626) => 627) FAL-PDU_req { 628) smpm_service_name = “SMPM_Send”, 629) arep_id = GetArepId (), 630) remote_address = FAL_Pdu_RemoteAddress (fal_pdu), 631) fda_pdu = BuildFAL-ErrPDU(request_pdu = fal_pdu, 632) error_class = “service”, 633) error_code = appropriate error code for detected error) 634) }	635) CLOSED
636) R3	637) CLOSED	638) FAL-PDU_ind 639) && EndpointType = “SERVER” 640) && FAL_Pdu_SvcType (fal_pdu) = “AR Establish Req” 641) && ConfigurationArCheckOK (fal_pdu) = False 642) => 643) FAL-PDU_req { 644) smpm_service_name = “SMPM_Send”, 645) arep_id = GetArepId (), 646) remote_address = FAL_Pdu_RemoteAddress (fal_pdu), 647) fal_pdu = BuildFAL-ErrPDU(request_pdu = fal_pdu, 648) error_class = “access”, 649) error_code = “config access already open”) 650) }	651) CLOSED
652) R4	653) CLOSED	654) FAL-PDU_ind 655) && EndpointType = “SERVER” 656) && FAL_Pdu_SvcType (fal_pdu) = “AR Establish Req” 657) && ConfigurationArCheckOK(fal_pdu) = True 658) => 659) RemoteAddress = FAL_Pdu_RemoteAddress () 660) EST_ind* { 661) arep_id = GetArepId (), 662) user_data = GetUserData (fal_pdu) 663) } 664) StartInactivityCloseTimer() 665) 666) * The server entity that receives this indication is responsible for performing parameter negotiations.	667) RESPON DING
668) R5	669) OPEN 670) RESPONDING 671) REQUESTING	672) FAL-PDU_ind 673) && FAL_Pdu_SvcType (fal_pdu) = “AR Establish Req” 674) => 675) FAL-PDU_req { 676) smpm_service_name = “SMPM_Send”, 677) arep_id = GetArepId (), 678) remote_address = FAL_Pdu_RemoteAddress (), 679) fal_pdu = BuildFAL-ErrPDU(request_pdu = fal_pdu, 680) error_class = “service”, 681) error_code = "object state conflict") 682) }	683) CLOSED
684)	685) REQUESTING	686) FAL-PDU_ind	693) CLOSED

#	Current state	Event or condition => action	Next state
R6		687) && FAL_Pdu_SvcType (fal_pdu) = "AR Establish Err" 688) => 689) EST_cnf(-) { 690) arep_id = GetArepId (), 691) user_data = FAL_Pdu_ServiceSpecificParameters(fal_pdu) 692) }	
694) R7	695) REQUESTING	696) FAL-PDU_ind 697) && FAL_Pdu_SvcType (fal_pdu) = "AR Establish Rsp" 698) => 699) BufferSize = FAL_Pdu_BufferSize(fal_pdu) 700) InactivityCloseTime = GetInactivityCloseTime(fal_pdu) 701) EST_cnf(+) { 702) arep_id = GetArepId (), 703) user_data = FAL_Pdu_ServiceSpecificParameters(fal_pdu) 704) } 705) StartInactivityCloseTimer()	706) OPEN
707) R8	708) RESPONDING 709) REQUESTING	710) FDA-PDU_ind 711) && FDA_Pdu_SvcType (fda_pdu) = "CS_ReqPDU" 712) => 713) FDA-PDU_req { 714) smpm_service_name = "SMPM_Send", 715) arep_id = GetArepId (), 716) remote_address = FDA_Pdu_RemoteAddress (fda_pdu), 717) fda_pdu = BuildFDA-ErrPDU(request_pdu = fda_pdu, 718) error_class = "service", 719) error_code = "object state conflict") 720) } 721) StartInactivityCloseTimer()	722) CLOSED
723) R9	724) RESPONDING	725) FAL-PDU_ind 726) && FAL_Pdu_SvcType (fal_pdu) = "ANY FAL PDU" 727) =>	728) CLOSED
729) R10	730) OPEN	731) FAL-PDU_ind 732) && FAL_Pdu_SvcType (fal_pdu) = "UCS_ReqPDU" 733) && FAL_Pdu_GetVcrId() <> NULL 734) => 735) UCS_ind { 736) arep_id = GetArepId (), 737) vcr_id = FAL_Pdu_GetVcrId(), 738) service_type = FAL_Pdu_SvcType(fal_pdu), 739) user_data = FAL_Pdu_ServiceSpecificParameters(fal_pdu) 740) } 741) StartInactivityCloseTimer()	742) OPEN
743) R11	744) OPEN	745) FAL-PDU_ind 746) && FAL_Pdu_SvcType (fal_pdu) = "CS_ReqPDU" 747) && FAL_Pdu_GetVcrId() = NULL 748) => 749) FAL-PDU_req { 750) smpm_service_name = "SMPM_Send", 751) arep_id = GetArepId (), 752) remote_address = FAL_Pdu_RemoteAddress (fal_pdu), 753) fal_pdu = BuildFAL-ErrPDU(request_pdu = fal_pdu, 754) error_class = "access", 755) error_code = "unrecognized FDA Address") 756) } 757) StartInactivityCloseTimer()	758) OPEN
759) R12	760) OPEN	761) FAL-PDU_ind 762) && FAL_Pdu_SvcType (fal_pdu) = "CS_ReqPDU" 763) => 764) CS_ind { 765) arep_id = GetArepId (), 766) vcr_id = FAL_Pdu_GetVcrId(), 767) service_type = FAL_Pdu_SvcType(fal_pdu), 768) user_data = FAL_Pdu_ServiceSpecificParameters(fal_pdu) 769) } 770) StartInactivityCloseTimer()	771) OPEN
772) R13	773) OPEN	774) FAL-PDU_ind 775) && FAL_Pdu_SvcType (fda_pdu) = "DTC_ReqPDU" 776) && MaxOutstandingReached() = True 777) => 778) FAL-PDU_req { 779) smpm_service_name = "SMPM_Send", 780) arep_id = GetArepId (),	787) OPEN

#	Current state	Event or condition => action	Next state
		781) remote_address = FAL_Pdu_RemoteAddress (fal_pdu), 782) fda_pdu = BuildFAL-ErrPDU(request_pdu = fal_pdu, 783) error_class = “resource”, 784) error_code = “max outstanding requests per session exceeded”) 785) } 786) StartInactivityCloseTimer()	
788) R14	789) OPEN	790) FAL-PDU_ind 791) && (FAL_Pdu_SvcType (fal_pdu) = “CS_RspPDU” 792) || FAL_Pdu_SvcType (fal_pdu) = “CS_ErrPDU) 793) => 794) CS_cnf { 795) arep_id = GetArepId (), 796) vcr_id = FAL_Pdu_GetVcrId(), 797) service_type = FAL_Pdu_SvcType(fal_pdu), 798) user_data = FAL_Pdu_ServiceSpecificParameters(fal_pdu) 799) } 800) StartInactivityCloseTimer()	801) OPEN
802) R15	803) OPEN REQUESTING 804) RESPONDING	805) ErrorToARPM 806) && ErrorType = “socket disconnect” 807) =>	808) CLOSED

4.8.2 ARPM

4.8.2.1 General

This ARPM is used by Publisher/Subscriber and Report Distribution ARs.

4.8.2.2 Primitive definitions

4.8.2.2.1 Primitives exchanged between ARPM and FSPM

Table 136 and Table 137 list the primitives exchanged between the ARPM and the FSPM.

Table 136 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
EST_req	FSPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
Abort_req	FSPM	vcr_id, abort_detail, abort_identifier, reason_code	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
UCS_req	FSPM	vcr_id, service_type, user_data	809) This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 137 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	vcr_id, abort_detail, abort_identifier, reason_code	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
UCS_ind	ARPM	vcr_id, service_type, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.

4.8.2.2.2 Parameters of FSPM/ARPM primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 138.

Table 138 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
vcr_id	This parameter is used to unambiguously identify an instance of the VCR that issued a primitive or that is the primitive destination. A means for such identification is not specified by this standard.
user_data	This parameter conveys AR-User data (Including the service specific parameters).
abort_detail	This parameter conveys the value that is used for the Abort_Detail parameter.
abort Identifier	This parameter conveys the value that is used for the Identifier parameter.
reason_code	This parameter conveys the value that is used for the Reason_Code parameter.
remote_address	This parameter conveys the remote address of the underlying layer.
service_type	This parameter conveys the protocol Id and the service within the protocol.

4.8.2.3 DLL mapping of MulticastAREP class

4.8.2.3.1 Formal definition

Subclause 4.8.2.3 describes the mapping of the Publisher / Subscriber and Report Distribution AREP Classes to the Socket model. The Socket mapping attributes and their permitted values used with the Publisher / Subscriber and Report Distribution AREP classes are defined in 4.8.2.3.

CLASS: **Multicast**
PARENT CLASS: AR Endpoint
ATTRIBUTES:

1	(m)	KeyAttribute:	LocalAndRemoteAddresses
1.1	(m)	KeyAttribute:	LocalAddress
1.2	(m)	KeyAttribute:	RemoteAddress
2	(m)	Attribute:	TransferType (CONN, CNLS)
4	(c)	Constraint:	Role = SUBSCRIBER | REPORT SINK
4.1	(m)	Attribute:	GuardBand
5	(c)	Constraint:	Role = PUBLISHER | REPORT SOURCE
5.1	(m)	Attribute:	TransmitDelayTime
6	(m)	Attribute:	BufferSize
7	(m)	Attribute:	HdrOptions
8	(c)	Constraint:	Role = SUBSCRIBER | REPORT SINK
8.1	(m)	Attribute:	MaxApduLength

4.8.2.3.2 Attributes

LocalAndRemoteAddresses

This key attribute identifies the AREP. It is composed of the local and remote addresses of the underlying layer for the AREP.

TransferType

This attribute specifies whether the connection-oriented (CONN) or connectionless (CNLS) services of the underlying layer are used by the AREP.

GuardBand

This conditional attribute specifies the lower and upper bound for the field being protected from rollover. It is present if the ROLE is SUBSCRIBER or REPORT SINK.

TransmitDelayTime

This conditional attribute specifies the amount of time that a sending session endpoint accumulates messages to send in its buffer. It is present if the ROLE is PUBLISHER or REPORT SOURCE. The time period starts when the first message is placed into the transmit buffer after either the buffer has been initialized or after it has been emptied by the previous transmission.

If the endpoint has no messages to send (the buffer is empty) and it receives a message to send, it loads the message into the buffer and waits for additional messages to concatenate into the buffer. If the buffer fills before the Transmit Delay Time has been reached, the Type 5 FAL AE stops the timer and sends the buffer. When this time interval has expired and the buffer has not yet filled, it sends the contents of the buffer.

BufferSize

This attribute specifies the size in octets of the session endpoint's send buffer. The buffer holds an integral number of messages. If a message is received for transfer that would overflow the buffer, the buffer is sent first, and the message is then added as the first message in the buffer.

HdrOptions

This attribute specifies which Message Trailer fields are present in the messages sent and received on the session.

MaxApduLength

This conditional attribute specifies the maximum permitted length in octets of APDUs sent and received on the AR. It is present if the ROLE is SUBSCRIBER or REPORT SINK.

4.8.2.4 MulticastARPM state machine

4.8.2.4.1 MulticastARPM states

The defined states and their descriptions of the Publisher / Subscriber ARPM are shown in Table 139 and Figure 3. For CONN, the closed state is never left until the connection has been established. How this connection is established is implementation dependent.

Table 139 – Publisher / subscriber ARPM states

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

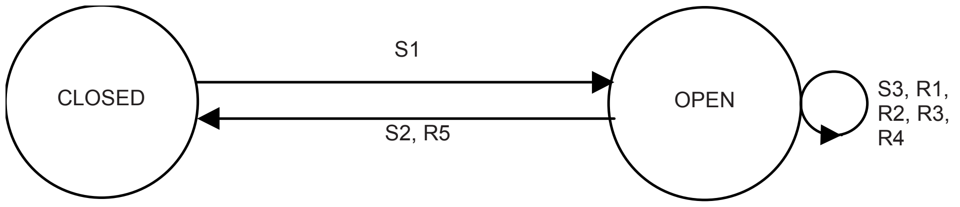

Figure 3 – State transition diagram of the publisher / subscriber ARPM

4.8.2.4.2 Multicast ARPM state table

Table 140 and Table 141 define the Multicast ARPM state machine.

Table 140 – MulticastARPM state table – sender transitions

#	Current state	Event or condition => action	Next state
810) S1	811) CLOSED	812) EST_req 813) => 814) EST_cnf(+){ 815) arep_id := GetArepId (), 816) user_data := “null” 817) }	818) OPEN
819) S2	820) OPEN 821)	822) Abort_req 823) =>	824) CLOSED
825) S3	826) OPEN	827) UCS_req 828) && Role = “PUBLISHER” || "REPORT SOURCE" 829) => 830) FAL-PDU_req { 831) dmpm_service_name := "SMPM_Send", 832) arep_id := GetArepId (), remote_address := RemoteAddress*, 833) fal_pdu := BuildFAL-ReqRspPDU (fal_service_type = service_type, 834) fal_data := user_data) 835) }	836) OPEN

Table 141 – MulticastARPM state table – receiver transitions

#	Current state	Event or condition => action	Next state
837) R1	838) OPEN	839) FAL-PDU_ind 840) && FAL_Pdu_Valid(fal_pdu) = False 841) => 842) Report to local management	843) OPEN
844) R2	845) OPEN	846) FAL-PDU_ind 847) && Endpoint Type = “SUBSCRIBER” || "REPORT SINK" 848) && FAL_Pdu_Confirmed(fal_pdu) = True 849) => 850) Report to local management	851) OPEN
852) R3	853) OPEN	854) FAL-PDU_ind 855) && Endpoint Type = “SUBSCRIBER” 856) && FAL_Pdu_GetVcrId() <> NULL 857) => 858) UCS_ind* { 859) arep_id = GetArepId (), 860) vcr_id = GetVcrId (), 861) service_type = GetServiceType(fal_pdu), 862) user_data = GetUserData(fal_pdu) 863) } 864) 865) * Delivered to each VCR that subscribes to the FDA Address in the APDU Header	866) OPEN
867) R4	868) OPEN	869) FAL-PDU_ind 870) && Endpoint Type = “REPORT SINK” 871) => 872) UCS_ind* { 873) arep_id = GetArepId (), 874) vcr_id = GetVcrId (), 875) service_type = GetServiceType(fal_pdu), 876) user_data = GetUserData(fal_pdu) 877) } 878) 879) * Delivered to all VCRs associated with this AREP	880) OPEN
881) R5	882) OPEN	883) ErrorToARPM 884) && ErrorType = “socket disconnect” 885) => 886) Abort_ind* { 887) arep_id = GetArepId (), 888) locally_generated = "True", 889) identifier = "FAL", 890) reason_code = "Socket disconnected" 891) } 892) 893) * Delivered to all VCR associated with this AREP	894) CLOSED

4.8.3 Functions used by ARPMs

Table 142 through Table 158 define the functions used by the ARPMs.

4.8.3.1 BuildFAL-ErrPDU()

Table 142 – BuildFAL-ErrPDU()

Function	
Returns a fal_pdu that is an error response with the specified errors. The request_pdu input parameter provides the necessary information to construct the appropriate error response pdu.	
Input parameters	Output parameters
request_pdu error_class error_code	fal_pdu

4.8.3.2 BuildFAL-ReqRspPDU()

Table 143 – BuildFAL-ReqRspPDU()

Function	
Builds an FAL PDU out of the parameters given as input variables. This includes: Building the FAL pdu header Adding the fal_data (AR user data) Generating the FAL-PDU trailer	
Input parameters	Output parameters
service_type fal_data	fal_pdu

4.8.3.3 GetArepId()

Table 144 – GetArepId()

Function	
Returns a value that can unambiguously identify the current AREP.	
Input parameters	Output parameters
None	AREP Identifier

4.8.3.4 ConfigurationArCheckOK()

Table 145 – ConfigurationArCheckOK()

Function	
Returns True if: The request fal_pdu is not a configuration request The request fal_pdu is a configuration request and there are no other configuration ARs established	
Input parameters	Output parameters
fal_pdu	True | False

4.8.3.5 FAL_Pdu_BufferSize()

Table 146 – FAL_Pdu_BufferSize()

Function	
This function returns the buffer size from the AR Establish fal_pdu.	
Input parameters	Output parameters
fal_pdu	BufferSize

4.8.3.6 FAL_Pdu_Confirmed()

Table 147 – FAL_Pdu_Confirmed()

Function	
This function returns true if the fal_pdu is confirmed	
Input parameters	Output parameters
fal_pdu	True | False

4.8.3.7 FAL_Pdu_DuplicateMsg ()

Table 148 – FAL_Pdu_DuplicateMsg ()

Function	
This function returns True if the received fal_pdu is determined to be a duplicate based on its message number and the vcr_id parameter. Otherwise, it returns False.	
Input parameters	Output parameters
vcr_id, fal_pdu	True | False

4.8.3.8 FAL_Pdu_GetVcrId()

Table 149 – FAL_Pdu_GetVcrId()

Function	
Using the FDA Address in the fal_pdu header, returns a value that identifies the destination VCR endpoint associated with the AR. May return a list if the AREP is a subscriber. Returns NULL if the FDA Address does not identify a VCR endpoint.	
Input parameters	Output parameters
fal_pdu	VCR Identifier

4.8.3.9 FAL_Pdu_InactivityCloseTime()

Table 150 – FAL_Pdu_InactivityCloseTime()

Function	
This function returns the InactivityCloseTime from the AR Establish fal_pdu.	
Input parameters	Output parameters
AR Establish fal_pdu	InactivityCloseTime

4.8.3.10 FAL_Pdu_TransmitDelayTime()

Table 151 – FAL_Pdu_TransmitDelayTime()

Function	
This function returns the TransmitDelayTime from the AR Establish fal_pdu.	
Input parameters	Output parameters
AR Establish fal_pdu	TransmitDelayTime

4.8.3.11 FAL_Pdu_SvcType()

Table 152 – FAL_Pdu_SvcType()

Function	
This function decodes the FAL-PDU that is conveyed in the fal_pdu parameter and retrieves the FAL-PDU service type.	
Input parameters	Output parameters
fal_pdu	service type

4.8.3.12 FAL_Pdu_RemoteAddress()

Table 153 – FAL_Pdu_RemoteAddress()

Function	
Returns the address of the sender of the received fal_pdu.	
Input parameters	Output parameters
fal_pdu	address of sender

4.8.3.13 FAL_Pdu_TrailerFields()

Table 154 – FAL_Pdu_TrailerFields()

Function	
This function returns the trailer fields of the fal_pdu.	
Input parameters	Output parameters
fal_pdu	Trailer fields present in the FAL-PDU.

4.8.3.14 FAL_Pdu_ServiceSpecificParameters()

Table 155 – FAL_Pdu_ServiceSpecificParameters()

Function	
This function returns the service specific parameters, including user data, from an fal_pdu.	
Input parameters	Output parameters
fal_pdu	Service Specific parameters

4.8.3.15 FAL_Pdu_Valid()

Table 156 – FAL_Pdu_Valid()

Function	
The function returns False if the ARPM can determine that: 1. the fal_pdu header or trailer contains invalid values or inconsistent values, or 2. the fal_pdu header or trailer contains options that do not match those negotiated for the AR, 3. the fal_pdu cannot be properly decoded or identified. The Service Specific parameters are not examined by this function.	
Input parameters	Output parameters
fal_pdu	True | False

4.8.3.16 MaxOutstandingReached()

Table 157 – MaxOutstandingReached()

Function	
The function tests the local counter for the current number of outstanding requests to determine if it has reached the limit defined by the MaxOutstanding session attribute. It returns True if the limit has been reached. If not, it increments the local counter for the current number of outstanding requests and returns False.	
Input parameters	Output parameters
None	True | False

4.8.3.17 StartInactivityCloseTimer()

Table 158 – StartInactivityCloseTimer()

Function	
The function (re)starts a timer for the period of time defined by the Inactivity Close Time.	
Input parameters	Output parameters
None	None

4.9 DLL mapping protocol machine (DMPM)

4.9.1 General

The DLL mapping is represented by the Socket model.

The Socket model defines an abstract service interface that permits the underlying layer to exist at the data link layer, network layer, or transport layer. Two types of services are supported by the Socket model, connection oriented (CONN) and connectionless (CNLS).

4.9.2 Primitive definitions

4.9.2.1 Primitives exchanged between DMPM and ARPM

Table 159 and Table 160 define the primitives exchanged between the DMPM and the ARPM.

Table 159 – Primitives issued by ARPM to DMPM

Primitive name	Source	Associated parameters	Functions
FAL-PDU_req	ARPM	smpm_service_name, remote_address, fal_pdu	This primitive is used request the DMPM to transfer an FAL-PDU. The list of smpm_service_names is: SMPM_Send

Table 160 – Primitives issued by DMPM to ARPM

Primitive name	Source	Associated parameters	Functions
FAL-PDU_ind	DMPM	smpm_service_name, remote_address, fal_pdu	This primitive is used to pass an FAL-PDU received as a Socket model service data unit to a designated ARPM. It also carries some of the Socket model parameters that are referenced in the ARPM. The list of smpm_service_names is: SMPM_Receive
ErrorToARPM	DMPM	originator, reason	This primitive is used to convey selected communication errors reported by the Socket model to a designated ARPM.

4.9.2.2 Parameters of ARPM/DMPM primitives

The parameters used with the primitives exchanged between the ARPM and the DMPM are described in Table 161.

Table 161 – Parameters used with primitives exchanged between ARPM and DMPM

Parameter name	Description
fal_pdu	This parameter conveys the value of the socket_user_data parameter.
smpm_service_name	This parameter conveys a Socket model service name.
remote_address	This parameter conveys the value of the remote address of the underlying layer.
originator	This parameter identifies the entity that detected the error that is reported using the ErrorToARPM primitive.
reason	This parameter identifies the cause of the error that is reported using the ErrorToARPM primitive.

4.9.2.3 Primitives exchanged between the socket model and DMPM

See Table 162 for the definitions.

NOTE 1 The following primitives and their parameters are exchanged between the Socket model and the DMPM.

NOTE 2 The DMPM instance uses either CNLS or CONN services.

NOTE 3 It is assumed that each ARPM is bound to the local address and therefore this is never an argument with any primitive.

Table 162 – Primitives exchanged between the socket model and DMPM

Primitive name	Source	Associated parameters
socket_send.ind	Socket model	socket_user_data
socket_sendto.ind	Socket model	socket_remote_address, socket_user_data
socket_sendto.req	DMPM	socket_remote_address, socket_user_data
socket_send.req	DMPM	socket_user_data

4.9.2.4 Parameters of DMPM/socket model primitives

See Table 163 for the definitions.

Table 163 – Parameters of DMPM/socket model primitives

Parameter name	Description
socket_remote_address	This parameter conveys the value of the remote Socket model address.
socket_user_data	The complete message that the socket model shall send to / received from specified destination

4.9.3 DMPM state machine

4.9.3.1 DMPM states

The defined state of the DMPM together with its description is listed in Table 164 and Figure 4.

Table 164 – DMPM state descriptions

State Name	Description
ACTIVE	The DMPM in the ACTIVE state is ready to transmit or receive primitives to or from the Socket model and the ARPM.

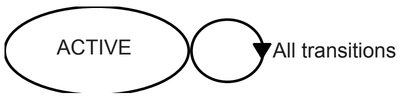

Figure 4 – State transition diagram of DMPM

4.9.3.2 DMPM state table

Table 165 and Table 166 define the DMPM state machine.

NOTE 1 Although each primitive contains all the available parameters, only those applicable to particular ARPM are relevant.

NOTE 2 Parameters starting with a capital letter, “TransmitDelayTime ” for instance, refer to those defined in the attribute list of each ARPM. Therefore, they are not conveyed by the service primitives defined here.

Table 165 – DMPM state table – sender transitions

#	Current state	Event or condition => action	Next state
895) S1	896) ACTI VE	897) FDA-PDU_req 898) && RemainingBufferSizeCheck (arep_id, fal_pdu) = "FIRST MESSAGE" 899) => 900) LoadBuffer(arep_id, fal_pdu) 901) StartTransmitDelayTimer(arep_id)	902) ACTIVE
903) S2	904) ACTI VE	905) FDA-PDU_req 906) && RemainingBufferSizeCheck (arep_id, fal_pdu) = "LESS THAN" 907) => 908) LoadBuffer(arep_id, fal_pdu)	909) ACTIVE
910) S3	911) ACTI VE	912) FDA-PDU_req 913) && ConnectionOriented() = True 914) && RemainingBufferSizeCheck (arep_id, fal_pdu) = "EQUAL" 915) => 916) socket_send.req { 917) user_data = GetBufferedData(arep_id) 918) }	919) ACTIVE
920) S4	921) ACTI VE	922) FDA-PDU_req 923) && ConnectionOriented() = FALSE 924) && RemainingBufferSizeCheck (arep_id, fal_pdu) = "EQUAL" 925) => 926) socket_sendto.req { 927) remote_address = remote_address, 928) user_data = GetBufferedData(arep_id) 929) }	930) ACTIVE
931) S5	932) ACTI VE	933) FDA-PDU_req 934) && ConnectionOriented() = True 935) && RemainingBufferSizeCheck (arep_id, fal_pdu) = "GREATER THAN" 936) => 937) socket_send.req { 938) user_data = fal_pdu 939) } 940) LoadBuffer(arep_id,fal_pdu) 941) StartTransmitDelayTimer(arep_id)	942) ACTIVE
943) S6	944) ACTI VE	945) FDA-PDU_req 946) &&ConnectionOriented()=FALSE 947) && RemainingBufferSizeCheck (arep_id, fal_pdu) = "GREATER THAN" 948) => 949) socket_sendto.req { 950) remote_address = remote_address, 951) user_data = GetBufferedData(arep_id) 952) } 953) LoadBuffer(arep_id, fal_pdu) 954) StartTransmitDelayTimer(arep_id)	955) ACTIVE
956) S7	957) ACTI VE	958) Transmit delay timer expires 959) && ConnectionOriented = False 960) => 961) socket_sendto.req { 962) remote_address = GetRemoteAddress(arep_id), 963) user_data = GetBufferedData(arep_id) 964) }	965) ACTIVE
966) S8	967) ACTI VE	968) Transmit delay timer expires 969) && ConnectionOriented() = True 970) => 971) socket_send.req { 972) user_data = GetBufferedData(arep_id) 973) }	974) ACTIVE

Table 166 – DMPM state table – receiver transitions

#	Current state	**Event** or **condition** **=> action**	Next state
975) R1	976) ACTIVE	977) socket_sendto.ind 978) => 979) FAL-PDU_ind { 980) smpm_service_name := "SMPM_Receive", 981) remote_address := socket_remote_address, 982) fal_pdu := socket_user_data 983) }	984) ACTIVE
985) R2	986) ACTIVE	987) socket_send.ind 988) => 989) FAL-PDU_ind { 990) smpm_service_name := "SMPM_Receive", remote_address := GetConnectionId(arep_id) , 991) fal_pdu := socket_user_data 992) }	993) ACTIVE
994) R3	995) ACTIVE	996) “Connection breaks” 997) => 998) ErrorToArpm { 999) originator= local_socket, 1000) reason = “Socket disconnected” 1001) }	1002) ACTIVE

4.9.3.3 Functions used by DMPM

Table 167 through Table 172 define the functions used by the DMPM.

4.9.3.3.1 Function ConnectionOriented

Table 167 – ConnectionOriented

Name	ConnectionOriented	Used in	ARPM
Input		Output	
		True | False	
Function			
1003) This function returns True if the SMPM instance is connection oriented (CONN) and False if the SMPM instance in connection less (CNLS).			

4.9.3.3.2 Function GetBufferedData

Table 168 – GetBufferedData

Function	
This function returns the contents of the AREP send buffer, and marks the buffer as empty.	
Input parameters	Output parameters
ArepId	buffered_data

4.9.3.3.3 Function GetConnectionId

Table 169 – GetConnectionId

Function	
This function returns the connection id of a connection-oriented socket.	
Input parameters	Output parameters
ArepId	remote_address

4.9.3.3.4 Function LoadBuffer

Table 170 – LoadBuffer

Function	
This function concatenates the user_data into the AREP buffer	
Input parameters	Output parameters
ArepId, user_data	None

4.9.3.3.5 Function RemainingBufferSizeCheck

Table 171 – RemainingBufferSizeCheck

Function	
This function compares the size of the user_data with the size of the space remaining in the AREP send buffer. It returns: "FIRST MESSAGE" the buffer is currently empty and the size of the user_data less than the size of the buffer. "EQUAL" the size of the user_data is the same size as the space remaining in the buffer. "LESS THAN" the size of the user_data is less than the size of the space remaining in the buffer. "GREATER THAN" the size of the user_data is greater than the size of the space remaining in the buffer.	
Input parameters	Output parameters
ArepId, user_data	buffer_status

4.9.3.3.6 StartTransmitDelayTimer

Table 172 – StartTransmitDelayTimer

Function	
This function starts or restarts the Transmit Delay Timer of the AREP identified by the ArepId parameter	
Input parameters	Output parameters
ArepId	None

Bibliography

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 61784-1, *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

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

ISO/IEC 8824:1990, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*[1]

____

[1] Withdrawn.

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

bsi.

...making excellence a habit.™