BS EN 61158-6-14:2014

BSI Standards Publication

Industrial communication networks — Fieldbus specifications

Part 6-14: Application layer protocol specification — Type 14 elements

bsi.

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 61158-6-14:2014. It is identical to IEC 61158-6-14:2014. It supersedes BS EN 61158-6-14:2012 which is withdrawn.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.

Published by BSI Standards Limited 2014

ISBN 978 0 580 79477 3
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 61158-6-14

October 2014

ICS 25.040.40; 35.100.70; 35.110

Supersedes EN 61158-6-14:2012

English Version

Industrial communication networks - Fieldbus specifications - Part 6-14: Application layer protocol specification - Type 14 elements (IEC 61158-6-14:2014)

Réseaux de communication industriels - Spécifications des bus de terrain - Partie 6-14: Spécification du protocole de la couche application - Eléments de type 14
(CEI 61158-6-14:2014)

Industrielle Kommunikationsnetze - Feldbusse - Teil 6-14: Protokollspezifikation des Application Layer (Anwendungsschicht) - Typ 14-Elemente
(IEC 61158-6-14:2014)

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.

CENELEC

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

Ref. No. EN 61158-6-14:2014 E

Foreword

The text of document 65C/764/FDIS, future edition 3 of IEC 61158-6-14, 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 61158-6-14:2014.

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 61158-6-14:2012.

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 61158-6-14:2014 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 61158-1	NOTE	Harmonized as EN 61158-1.
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 61158-3-14	-	Industrial communication networks - Fieldbus specifications - Part 3-14: Data-link layer service definition - Type 14 elements	EN 61158-3-14	-
IEC 61158-4-14	-	Industrial communication networks - Fieldbus specifications - Part 4-14: Data-link layer protocol specification - Type 14 elements	EN 61158-4-14	-
IEC 61158-5-14	-	Industrial communication networks - Fieldbus specifications - Part 5-14: Application layer service definition - Type 14 elements	EN 61158-5-14	-
IEC 61158-6	series	Industrial communication networks - Fieldbus specifications - Part 6: Application layer protocol specification	EN 61158-6	series
ISO/IEC 646	-	Information technology - ISO 7-bit coded character set for information interchange	-	-
ISO/IEC 2375	-	Information technology - Procedure for registration of escape sequences and coded character sets	-	-
ISO/IEC 7498-1	-	Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model	-	-
ISO/IEC 8802-3	-	Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications	-	-

Publication	Year	Title	EN/HD	Year
ISO/IEC 8822	-	Information technology - Open Systems Interconnection - Presentation service definition	-	-
ISO/IEC 8824	1990	Information technology - Open Systems Interconnection - Specification of 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	-	-
ISO/IEC/IEEE 60559	-	Information technology - Microprocessor Systems - Floating-Point arithmetic	-	-
IEEE 754	-	IEEE Standard for Floating-Point Arithmetic	-	-

CONTENTS

INTRODUCTION 9
1 Scope 10
1.1 General 10
1.2 Specifications 10
1.3 Conformance 11
2 Normative references 11
3 Terms, definitions, symbols, abbreviations and conventions 12
3.1 Referenced terms and definitions 12
3.2 Fieldbus application layer specific terms and definitions 13
3.3 Abbreviations and symbols 15
3.4 Conventions 17
4 Abstract syntax 18
4.1 Fixed format PDU description 18
4.2 Object definitions in FAL management ASE 27
4.3 Definition of objects used in Type 14 application access entity 33
5 Transfer syntax 36
5.1 Encoding of basic data types 36
5.2 Encoding of Type 14 APDU header 42
5.3 Encoding of FAL management entity service parameters 43
5.4 Encoding of AAE Services 49
6 Structure of FAL protocol state machines 58
7 AP-Context state machine 59
7.1 Primitives exchanged between ALU and ALE 59
7.2 Protocol state machine descriptions 59
7.3 State transitions 60
7.4 Function descriptions 66
8 FAL management state machines 66
8.1 Primitives 66
8.2 Protocol state machine descriptions 67
8.3 State transitions 68
8.4 Function descriptions 70
9 Application access entity protocol machine 74
9.1 Primitives 74
9.2 AAE state machine 76
9.3 Event ASE protocol machine 78
9.4 Domain ASE protocol machine 79
9.5 Block ASE protocol machine 83
10 Application relationship state machine 85
10.1 Primitives 85
10.2 AREP state description 87
10.3 State transitions 87
10.4 Function descriptions 88
11 DLL mapping protocol machine 88
11.1 Concept 88

11.2 Primitives 89
11.3 State description 89
11.4 State transitions 89
11.5 Function description 90
Bibliography 91

Figure 1 – State transition diagram 17
Figure 2 – Exchanged primitives of protocol state machine 59
Figure 3 – ACE protocol state machine 60
Figure 4 – FME protocol state machine 68
Figure 5 – AAE state transition diagrams 76
Figure 6 – Event ASE state transition diagrams 78
Figure 7 – Domain ASE state transition diagram 80
Figure 8 – Block ASE state transition diagrams 84
Figure 9 – AREP state transition diagrams 87
Figure 10 – ESME state transition 90

Table 1 – State machine description elements 17
Table 2 – Definition of Type 14 MOB header object 27
Table 3 – Definition of Type 14 device descriptor object 27
Table 4 – Definition of the time synchronization object 28
Table 5 – Definition of maximum response time object 28
Table 6 – Definition of the Type 14 communication scheduling management object 29
Table 7 – Definition of the device application information object 29
Table 8 – Definition of FB application information header 29
Table 9 – Definition of domain application information header 30
Table 10 – Definition of Type 14 link object header 30
Table 11 – Definition of Type 14 FRT link object header 31
Table 12 – Definition of FB application information object 31
Table 13 – Definition of Type 14 link object 31
Table 14 – Definition of Type 14 FRT link object 32
Table 15 – Definition of domain application information object 33
Table 16 – Definition of domain object 33
Table 17 – Definition of simple variable object 34
Table 18 – Definition of event object 34
Table 19 – Definition of Type 14 socket mapping object 35
Table 20 – Definition of Type 14 socket timer object 35
Table 21 – Definition of ErrorType object 36
Table 22 – Encoding of Boolean value TRUE 36
Table 23 – Encoding of Boolean value FALSE 36
Table 24 – Encoding of Unsigned8 data type 37
Table 25 – Encoding of Unsigned16 data type 37
Table 26 – Encoding of Unsigned32 data type 37
Table 27 – Encoding of Unsigned64 data type 37

Table 28 – Encoding of Int8 data type 38
Table 29 – Encoding of Int16 data type 38
Table 30 – Encoding of Int32 data type 38
Table 31 – Encoding of Int64 data type 39
Table 32 – Encoding of Real type 39
Table 33 – Encoding of VisibleString data type 39
Table 34 – Encoding of OctetString data type 40
Table 35 – Encoding of BitString data type 40
Table 36 – Encoding of TimeOfDay data type 40
Table 37 – Encoding of BinaryDate data type 41
Table 38 – Encoding of PrecisionTimeDifference data type 42
Table 39 – Encoding of Type 14 application layer service message header 42
Table 40 – Encoding of EM_DetectingDevice request parameters 43
Table 41 – Encoding of EM_OnlineReply request parameters 43
Table 42 – Encoding of EM_GetDeviceAttribute request parameters 44
Table 43 – Encoding of EM_GetDeviceAttribute positive response parameters 44
Table 44 – Encoding of EM_GetDeviceAttribute negative response parameters 45
Table 45 – Encoding of EM_ActiveNotification request parameters 46
Table 46 – Encoding of EM_ConfiguringDevice request parameters 47
Table 47 – Encoding of EM_ConfiguringDevice positive response parameters 48
Table 48 – Encoding of EM_ConfiguringDevice negative response parameters 48
Table 49 – Encoding of EM_SetDefaultValue request parameters 48
Table 50 – Encoding of EM_SetDefaultValue positive response parameters 48
Table 51 – Encoding of clear device attribute service refuse packet 49
Table 52 – Encoding of DomainDownload request parameters 49
Table 53 – Encoding of domain download service response packet 49
Table 54 – Encoding of DomainDownload negative response parameters 49
Table 55 – Encoding of DomainUpload request parameters 50
Table 56 – Encoding of DomainUpload positive response parameters 50
Table 57 – Encoding of DomainUpload negative response parameters 50
Table 58 – Encoding of EventRoport request parameters 51
Table 59 – Encoding of EventRoportAcknowledge request parameters 51
Table 60 – Encoding of EventRoportAcknowledge positive response parameters 51
Table 61 – Encoding of EventRoportAcknowledge negative response parameters 51
Table 62 – Encoding of ReportConditionChanging request parameters 52
Table 63 – Encoding of ReportConditionChanging positive response parameters 52
Table 64 – Encoding of ReportConditionChanging negative response parameters 52
Table 65 – Encoding of Read request parameters 52
Table 66 – Encoding of Read positive response parameters 53
Table 67 – Encoding of Read negative response parameters 53
Table 68 – Encoding of Write request parameters 53
Table 69 – Encoding of Write positive response parameters 53
Table 70 – Encoding of Write negative response parameters 54

Table 71 – Encoding of VariableDistribute request parameters 54
Table 72 – Encoding of FRTRead request parameters 54
Table 73 – Encoding of FRTRead positive response parameters 54
Table 74 – Encoding of FRTRead negative response parameters 55
Table 75 – Encoding of FRTWrite request parameters 55
Table 76 – Encoding of FRTWrite positive response parameters 55
Table 77 – Encoding of FRTWrite negative response parameters 55
Table 78 – Encoding of FRTVariableDistribute request parameters 56
Table 79 – Encoding of BlockTransmissionOpen request parameters 56
Table 80 – Encoding of BlockTransmissionOpen positive response parameters 56
Table 81 – Encoding of BlockTransmissionOpen negative response parameters 56
Table 82 – Encoding of BlockTransmissionClose request parameters 57
Table 83 – Encoding of BlockTransmissionClose positive response parameters 57
Table 84 – Encoding of BlockTransmissionClose negative response parameters 57
Table 85 – Encoding of BlockTransmit request parameters 57
Table 86 – Encoding of BlockTransmissionHeartbeat request parameters 58
Table 87 – Primitives delivered by ALU to ALE 59
Table 88 – Primitives delivered by ALE to ALU 59
Table 89 – ACE state descriptions 60
Table 90 – ACE state transitions (sender) 60
Table 91 – ACE state transitions (receiver) 63
Table 92 – APServiceType() descriptions 66
Table 93 – Primitives delivered by application layer user to FME 66
Table 94 – Primitives delivered by FME to application layer user 66
Table 95 – Primitive parameters exchanged between FME and application layer user 67
Table 96 – Primitives delivered by FME to ESME 67
Table 97 – Primitives delivered by ESME to FME 67
Table 98 – Primitives parameters exchanged between FME and ESME 67
Table 99 – State transitions of Type 14 FME 68
Table 100 – RcvNewIpAddress() descriptions 70
Table 101 – Attribute_Set() descriptions 71
Table 102 – RestoreDefaults() descriptions 71
Table 103 – NewAddress() descriptions 71
Table 104 – Restart_Type 14RepeatTimer() descriptions 71
Table 105 – Clear_DuplicatePdTagFlag() descriptions 71
Table 106 – Type 14RepeatTimerExpire() descriptions 72
Table 107 – Send_EM_ReqRspMessage() descriptions 72
Table 108 – Send_EM_CommonErrorRsp() descriptions 72
Table 109 – SntpSyncLost() descriptions 72
Table 110 – IPAddressCollision() descriptions 73
Table 111 – RecvMsg() descriptions 73
Table 112 – QueryMatch() descriptions 73
Table 113 – MessageIDMatch() descriptions 73

Table 114 – DevId_Match() descriptions 73
Table 115 – PdTag_Match() descriptions 74
Table 116 – Set_Attribute_Data() descriptions 74
Table 117 – Set_DuplicatePdTagFlag() descriptions 74
Table 118 – Primitives issued by ALU to AAE 74
Table 119 – Primitives issued by AAE to ALU 75
Table 120 – Primitives parameters exchanged between AAE and ALU 75
Table 121 – Primitives issued by AAE to ESME 75
Table 122 – Primitives issued by ESME to AAE 75
Table 123 – Primitive parameters exchanged between AAE and ESME 76
Table 124 – AAE state descriptions 76
Table 125 – AAE state transitions (sender) 76
Table 126 – AAE state transitions (receiver) 77
Table 127 – ServiceType() descriptions 78
Table 128 – State value of event management 78
Table 129 – Event ASE state transition table 79
Table 130 – Domain state value 79
Table 131 – Domain ASE state transition table 80
Table 132 – Domain_DownloadSucceed() description 82
Table 133 – Domain_WriteBuffer() description 83
Table 134 – IncreamentInvokeDomainCounter() description 83
Table 135 – DecreamentInvokeDomainCounter() description 83
Table 136 – State value of Block transmission 83
Table 137 – Block ASE state transition table 84
Table 138 – BlockTransmissionOpenSucceed() descriptions 85
Table 139 – BlockTransmissionCloseSucceed() descriptions 85
Table 140 – ReceiveBlockTransmissionHeartbeat_timeout() description 85
Table 141 – Primitives issued by FME(or AAE) to AREP 86
Table 142 – Primitives issued by AREP to FME(or AAE) 86
Table 143 – Primitives parameters exchanged between AREP and FME(or AAE) 86
Table 144 – Primitives issued by AREP to ESME 86
Table 145 – Primitives issued by ESME to AREP 86
Table 146 – Primitive parameters exchanged between AREP and ESME 87
Table 147 – AREP state descriptions 87
Table 148 – AREP state transitions 87
Table 149 – AREPType() descriptions 88
Table 150 – ServiceType() descriptions 88
Table 151 – The primitives exchanged between transport layer and ESME 89
Table 152 – Primitives parameters exchanged between Transport Layer and ESME 89
Table 153 – ESME state description 89
Table 154 – ECFME state transitions 90
Table 155 – ServiceType()description 90

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 6-14: Application layer protocol specification – Type 14 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 14 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 specifies interactions between remote applications and defines the externally visible behavior provided by the Type 14 fieldbus application layer in terms of

a) the formal abstract syntax defining the application layer protocol data units conveyed between communicating application entities;
b) the transfer syntax defining encoding rules that are applied to the application layer protocol data units;
c) the application context state machine defining the application service behavior visible between communicating application entities;
d) the application relationship state machines defining the communication behavior visible between communicating application entities.

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

a) define the wire-representation of the service primitives defined in IEC 61158-5-14, and
b) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the Type 14 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI application layer structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-14.

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 the 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 61158-3-14, *Industrial communication networks – Fieldbus specifications – Part 3-14: Data-link layer service definition – Type 14 elements*

IEC 61158-4-14, *Industrial communication networks – Fieldbus specifications – Part 4-14: Data-link layer protocol specification – Type 14 elements*

IEC 61158-5-14, *Industrial communication networks – Fieldbus specifications – Part 5-14: Application layer service definition – Type 14 elements*

IEC 61158-6 (all parts)*, Industrial communication networks – Fieldbus specifications – Part 6: Application layer protocol specification*

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

ISO/IEC 2375, *Information technology – Procedure for registration of escape sequences and coded character sets*

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

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

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

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

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

[1] Withdrawn.

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

ISO/IEC/IEEE 60559, *Information technology – Microprocessor Systems – Floating-Point arithmetic*

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

3 Terms, definitions, symbols, abbreviations and conventions

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

3.1 Referenced terms and definitions

3.1.1 ISO/IEC 7498-1 terms

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

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

3.1.2 ISO/IEC 8822 terms

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

a) abstract syntax
b) presentation context

3.1.3 ISO/IEC 9545 terms

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

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

3.1.4 ISO/IEC 8824 terms

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

a) object identifier

b) type

3.1.5 Fieldbus data-link Layer terms

For the purposes of this document, the following terms as defined in IEC 61158-3-14 and IEC 61158-4-14 apply.

a) DL-Time

b) DL-Scheduling-policy

c) DLCEP

d) DLC

e) DL-connection-oriented mode

f) DLPDU

g) DLSDU

h) DLSAP

i) link

j) network address

k) node address

l) node

m) scheduled

3.2 Fieldbus application layer specific terms and definitions

3.2.1
access control
control on the reading and writing of an object

3.2.2
access path
association of a symbolic name with a variable for the purpose of open communication

3.2.3
communication macrocycle
set of basic cycles needed for a configured communication activity in a macro network segment

3.2.4
communication scheduling
algorithms and operation for data transfers occurring in a deterministic and repeatable manner

3.2.5
configuration (of a system or device)
step in system design: selecting functional units, assigning their locations and defining their interconnections

3.2.6
cyclic
repetitive in a regular manner

3.2.7
destination FB Instance
FB instance that receives the specified parameters

3.2.8
domain
part of memory used to store code or data

3.2.9
domain download
operation to write data in a domain

3.2.10
domain upload
operation to read data from a domain

3.2.11
entity
particular thing, such as a person, place, process, object, concept, association, or event

3.2.12
Type 14 bridge
DL-relay entity which performs synchronization between links (buses) and may perform selective store-and-forward and routing functions to connect two micro network segments

3.2.13
identifier
16-bit word associated with a system variable

3.2.14
index
address of an object within an application process

3.2.15
instance
actual physical occurrence of an object within a class that identifies one of many objects within the same object class

3.2.16
instantiation
creation of an instance of a specified type

3.2.17
management information
network-visible information for the purpose of managing the field system

3.2.18
management information base
organized list of management information

3.2.19
mapping
set of values having defined correspondence with the quantities or values of another set

3.2.20
member
piece of an attribute that is structured as an element of an array

3.2.21
message filtering
decision on a message according to a special rule

3.2.22
micro segment
part of a network, where special scheduling is implemented

3.2.23
offset
number of octets from a specially designated position

3.2.24
phase
elapsed fraction of a cycle, measured from some fixed origin

3.2.25
process interface
data exchange and information mapping between physical process and application unit

3.2.26
real-time
ability of a system to provide a required result in a bounded time

3.2.27
real-time communication
transfer of data in real-time

3.2.28
Real-Time Ethernet
RTE
ISO/IEC 8802-3-based network that includes real-time communication

Note 1 to entry: Other communication can be supported, providing the real-time communication is not compromised.

Note 2 to entry: This definition is dedicated, but not limited, to ISO/IEC 8802-3. It could be applicable to other IEEE 802 specifications, for example IEEE 802.1Q.

3.2.29
schedule
temporal arrangement of a number of related operations

3.2.30
scheduling macrocycle
time interval to implement a specific schedule

3.2.31
source FB Instance
FB instance that sends a specific parameter

3.2.32
time offset
time difference from a specially designated time

3.3 Abbreviations and symbols

AAE	Application Access Entity
AE	Application Entity
AL	Application Layer
ALE	Application Layer Entity
ALP	Application Layer Protocol

APO	Application Object
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Identifier
AR	Application Relationship
ARP	Address Resolution Protocol
AREP	Application Relationship End Point
ASE	Application Service Element
Cnf	Confirmation
CR	Communication Relationship
CREP	Communication Relationship End Point
CSMA/CD	Carrier Sense Multiple Access Protocol with Collision Detection
DD	Device Description
DHCP	Dynamic Host Configuration Protocol
DL-	(as a prefix) data-link-
DLCEP	Data-link Connection End Point
DLL	Data-link Layer
DLE	Data-link Entity
DLM	Data-link-management
DLS	Data-link Service
DLSAP	Data-link Service Access Point
DLSDU	DL-service-data-unit
ECSME	Type 14 communication scheduling management entity
EM_	(as a prefix) Type 14 Management
ESME	Type 14 Socket Mapping Entity
FB	Function Block
FBAP	Function Block Application Process
FME	FAL Management Entity
FRT	Fast Real-time
Ind	Indication
IP	Internet Protocol
LLC	Logical Link Control
LMP	Link Management Protocol
MAC	Medium Access Control
MAU	Medium Attachment Unit
MOB	Management Object Base
PAD	Pad (bits)
PDU	Protocol Data Unit
P/S	Publisher/Subscriber
Req	Request
Rsp	Response
RTE	Real-Time Ethernet
RT-Ethernet	Real-Time Ethernet
SAP	Service Access Point
SDU	Service Data Unit
SME	System Management Entity

SNTP	Simple Network Time Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
.cnf	Confirm Primitive
.ind	Indication Primitive
.req	Request Primitive
.rsp	Response Primitive

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 61158-5-14. 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 61158-5-14. 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 state machines for Type 14

A state machine describes the state sequence of an entity and can be represented by a state transition diagram and/or a state table.

In a state transition diagram (Figure 1), the transition between two states represented by circles is illustrated by an arrow beside which the transition events or conditions are presented.

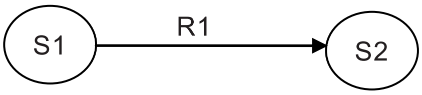

Figure 1 – State transition diagram

Table 1 – State machine description elements

#	Current state	Events or conditions that trigger this state transaction => 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 transition table (Table 1) are as follows.

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

xxx A parameter name.

Example:

Identifier := reason
means value of a 'reason' parameter is assigned to a parameter called 'Identifier.'
"xxx" Indicates fixed value.

Example:

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

= A logical condition to indicate an item on the left is equal to an item on the right.
< A logical condition to indicate an item on the left is less than the item on the right.
> A logical condition to indicate an item on the left is greater than the item on the right.
<> A logical condition to indicate an item on the left is not equal to an item on the right.
&& Logical "AND"
|| Logical "OR"

Service.req represents a Request Primitive; Service.req{} indicates that a request primitive is sent;

Service.ind represents an Indication Primitive; Service.ind{} indicates that an Indication Primitive is received;

Service.rsp represents a Response Primitive; Service.rsp{} indicates that a Response Primitive is sent;

Service.cnf represents a Confirm Primitive; Service.cnf{} indicates that a Confirm Primitive is received.

4 Abstract syntax

4.1 Fixed format PDU description

Type 14 PDU consists of fixed-length PDU header and variable-length PDU body. The former contains service type, message type and message length, etc.

```
Type 14 PDU : : = CHOICE {
          confirmed-RequestPDU            [0]     IMPLICIT  Confirmed-RequestPDU,
          confirmed-ResponsePDU           [1]     IMPLICIT  Confirmed-ResponsePDU,
          confirmed-ErrorPDU              [2]     IMPLICIT  Confirmed-ErrorPDU,
          unconfirmed-RequestPDU          [3]     IMPLICIT  Unconfirmed-RequestPDU
}
Confirmed-RequestPDU : : = SEQUENCE {
          pduHeader                  PDUHeader,
          confirmed-request          Confirmed-Request
}
Confirmed-ResponsePDU : : = SEQUENCE {
          pduHeader                  PDUHeader,
          confirmed-response         Confirmed-Response
}
Confirmed-ErrorPDU : : = SEQUENCE {
          pduHeader                  PDUHeader,
          confirmed-error            Confirmed-Error
}
Unconfirmed-RequestPDU : : = SEQUENCE {
          pduHeader                  PDUHeader,
          unconfirmed-request        Unconfirmed-Request
}
```

4.1.1 Confirmed request service

```
Confirmed- Request : : = CHOICE {
        EM_GetDeviceAttribute       [0]     IMPLICIT  EM_GetDeviceAttribute-RequestPDU,
        EM_ConfiguringDevice        [1]     IMPLICIT  EM_ConfiguringDevice-RequestPDU,
        EM_SetDefaultValue          [2]     IMPLICIT  EM_SetDefaultValue-RequestPDU,
        DomainDownload              [3]     IMPLICIT  DomainDownload-RequestPDU,
        DomainUpload                [4]     IMPLICIT  DomainUpload-RequestPDU,
        AcknowledgeEventRoport      [5]     IMPLICIT  AcknowledgeEventNotifi-RequestPDU,
        ReportConditionChanging     [6]     IMPLICIT  AlterEventConditionMon-RequestPDU,
        Read                        [7]     IMPLICIT  Read-RequestPDU,
        Write                       [8]     IMPLICIT  Write-RequesPDU,
        FRTRead                     [9]     IMPLICIT  FRTRead-RequestPDU,
        FRTWrite                    [10]    IMPLICIT  FRTWrite-RequesPDU
        BlockTransmissionOpen       [11]    IMPLICIT  OpenBlockTransmission-RequestPDU,
        BlockTransmissionClose      [12]    IMPLICIT  CloseBlockTransmission-RequestPDU,
}
```

4.1.2 Confirmed response service

```
Confirmed- Response : : = CHOICE {
        EM_GetDeviceAttribute       [0] IMPLICIT  EM_GetDeviceAttribute-ResponsePDU,
        EM_ConfiguringDevice        [1] IMPLICIT  EM_ConfiguringDevice-ResponsePDU,
        EM_SetDefaultValue          [2] IMPLICIT  EM_SetDefaultValue-ResponsePDU,
        DomainDownload              [3] IMPLICIT  DomainDownload-ResponsePDU,
        DomainUpload                [4] IMPLICIT  DomainUpload-ResponsePDU,
        AcknowledgeEventRoport      [5] IMPLICIT  AcknowledgeEventNotifi-ResponsePDU,
        ReportConditionChanging     [6] IMPLICIT  AlterEventConditionMon-ResponsePDU,
        Read                        [7] IMPLICIT  Read-ResponsePDU,
        Write                       [8] IMPLICIT  Write-ResponsePDU,
        FRTRead                     [9] IMPLICIT  FRTRead-ResponsePDU,
        FRTWrite                    [10] IMPLICIT  FRTWrite-ResponsePDU
        BlockTransmissionOpen       [11] IMPLICIT  OpenBlockTransmission-ResponsePDU,
        BlockTransmissionClose      [12] IMPLICIT  CloseBlockTransmission-ResponsePDU,
}
```

4.1.3 Confirmed error

```
Confirmed- Error : : = CHOICE {
        EM_GetDeviceAttribute       [0]     IMPLICIT  Error-Type,
        EM_ConfiguringDevice        [1]     IMPLICIT  Error-Type,
        EM_SetDefaultValue          [2]     IMPLICIT  Error-Type,
        DomainDownload              [3]     IMPLICIT  Error-Type,
        DomainUpload                [4]     IMPLICIT  Error-Type,
        AcknowledgeEventRoport      [5]     IMPLICIT  Error-Type,
        ReportConditionChanging     [6]     IMPLICIT  Error-Type,
        Read                        [7]     IMPLICIT  Error-Type,
        Write                       [8]     IMPLICIT  Error-Type,
        FRTRead                     [9]     IMPLICIT  Error-Type,
        FRTWrite                    [10]    IMPLICIT  Error-Type
        BlockTransmissionOpen       [11]    IMPLICIT  Error-Type
        BlockTransmissionClose      [12]    IMPLICIT  Error-Type
}
```

4.1.4 Error type

```
ErrorType : : = SEQUENCE {
        ErrorClass                  [0]            IMPLICIT  Integer8,
        ErrorCode                   [1]            IMPLICIT  Integer8,
        AdditionalCode              [2]            IMPLICIT  Integer8,
        Reserved                    [3]            IMPLICIT  OctetString,
        AdditionalDescription       [4]            IMPLICIT  VisibleString
}
```

4.1.5 Error class

```
ErrorClass ::=  CHOICE {                                                    ErrorCode
     Resource                         [0]   IMPLICIT  Integer8 {
                                                 memory-unavailable         (0),
                                                 Other                               (1)
     },
     Service                         [1]   IMPLICIT  Integer8 {
                                                 object-state-conflict               (0),
                                                 object-constraint-conflict          (1),
                                                 parameter-inconsistent              (2),
                                                 illegal-parameter                   (3),
                                                 Size Error                          (4),
                                                 Other                               (5)
     }
     Access                          [2]   IMPLICIT  Integer8 {
                                                 object-access-unsupported           (0),
                                                 object-non-existent                 (1),
                                                 object-access-denied                (2),
                                                 hardware-fault                      (3),
                                                 type-conflict                       (4),
                                                 object-attribute-inconsistent       (5),
                                                 Access-to-element-unsupported       (6),
                                                 Other                               (7)
     }
     Timer                           [3]   IMPLICIT  Integer8 {
                                                 Timer-Expire                        (0),
                                                 Timer-Error                         (1),
                                                 Other                               (2)
     },
     Other                           [4]   IMPLICIT  Integer8 {
                                                 Other                               (0)
     }
}
```

4.1.6 Unconfirmed request

```
Unconfirmed-Request: : = CHOICE {
        EM_DetectingDevice            [0]     IMPLICIT  EM_DetectingDevice-RequestPDU,
        EM_OnlineReply                [1]     IMPLICIT  EM_OnlineReply-RequestPDU,
        EM_ActiveNotification         [2]     IMPLICIT  EM_ActiveNotification-RequestPDU,
        EventRoport                   [3]     IMPLICIT  EventRoport-RequestPDU,
        VariableDistribute            [4]     IMPLICIT  VariableDistribute-RequestPDU,
        FRTVariableDistribute         [5]     IMPLICIT  FRTVariableDistribute-RequestPDU
        BlockTransmit                 [6]     IMPLICIT  BlockTransmit-RequestPDU
        BlockTransmissionHeartbeat[7]         IMPLICIT  RequestPDU-RequestPDU
}
```

4.1.7 Type 14 application layer PDU

```
ApplicationLayerPDU ::=  SEQUENCE {
        PDUHeader,
        PDUBody             CHOICE {
                    Confirmed-Request,
                    Confirmed-Response,
                    Confirmed-Error,
                    Unconfirmed-Request
        }
}
```

4.1.8 APDU header format

```
PDUHeader : : = SEQUENCE {
        ServiceID                         [0]     IMPLICIT  Unsigned8,
        Reserved                          [1]     IMPLICIT  OctetString,
        Length                            [2]     IMPLICIT  Unsigned16,
        MessageID                         [3]     IMPLICIT  Unsigned16
}
```

4.1.9 FAL Management Entity services

4.1.9.1 EM_DetectingDevice service

```
EM_DetectingDevice-RequestPDU : : = SEQUENCE {
        QueryType                         [0]     IMPLICIT  Unsigned8,
        Reserved                            [1]      IMPLICIT  OctetString,
        PDTag                               [2]      IMPLICIT  VisibleString,
        FBTag                               [3]      IMPLICIT  VisibleString,
        ElementID                           [4]      IMPLICIT  Unsigned16
}
```

4.1.9.2 EM_OnlineReply service

```
EM_OnlineReply -RequestPDU : : = SEQUENCE {
          QueryType                           [0]      IMPLICIT  Unsigned8,
          DuplicateTagDetected                [1]      IMPLICIT  Boolean,
          Reserved                            [2]      IMPLICIT  OctetString,
          QueriedObjectIpAddress              [3]      IMPLICIT  Unsigned32,
          QueriedObjectDeviceID               [4]      IMPLICIT  VisibleString,
          QueriedObjectPDTag                  [5]      IMPLICIT  VisibleString
}
```

4.1.9.3 EM_GetDeviceAttribute service

```
EM_GetDeviceAttribute-RequestPDU : : = SEQUENCE {
          DestinationIPAddress                [0]      IMPLICIT  Unsigned32,
}
EM_GetDeviceAttribute-ResponsePDU : : = CHOICE {
          EM_GetDeviceAttribute-PositiveResponsePDU,
          EM_GetDeviceAttribute-NegativeResponsePDU
}
EM_GetDeviceAttribute-PositiveResponsePDU : : = SEQUENCE {
          DeviceID                            [0]      IMPLICIT  VisibleString,
          PdTag                               [1]      IMPLICIT  VisibleString,
          Status                              [2]      IMPLICIT  Unsigned8,
          DeviceType                          [3]      IMPLICIT  Unsigned8,
          Annunciation Interval               [4]      IMPLICIT  Unsigned16,
          Annunciation Version Number         [5]       IMPLICIT  Unsigned16,
          DuplicateTagDetected                [6]      IMPLICIT  Boolean,
          DeviceRedundancyNumber              [7]      IMPLICIT  Unsigned8,
          LANRedundancyPort                   [8]      IMPLICIT  Unsigned16
          DeviceRedundancy State              [9]      IMPLICIT  Unsigned8,
          MaxRedundancyNumber                 [10]     IMPLICIT  Unsigned8,
          ActiveIPAddress                     [11]     IMPLICIT  Unsigned32
}
EM_GetDeviceAttribute-NegativeResponsePDU : : = SEQUENCE {
          DestinationIPAddress                [0]      IMPLICIT       Unsigned32,
          Error-Type                          [1]      IMPLICIT       Error-Type
}
```

4.1.9.4 EM_ActiveNotification service

```
EM_ActiveNotification-RequestPDU : : = SEQUENCE {
          DeviceID                            [0]      IMPLICIT  VisibleString,
          PdTag                               [1]      IMPLICIT  VisibleString,
          Status                              [2]      IMPLICIT  Unsigned8,
          DeviceType                          [3]      IMPLICIT  Unsigned8,
          AnnunciationVersionNumber           [4]      IMPLICIT  Unsigned16,
          Device Redundancy Number            [5]      IMPLICIT  Unsigned8,
          DeviceRedundancyState               [6]      IMPLICIT  Unsigned8,
          LANRedundancyPort                   [7]      IMPLICIT  Unsigned16
          DuplicateTagDetected                [8]      IMPLICIT  Boolean,
          MaxRedundancyNumber                 [9]      IMPLICIT  Unsigned8,
          Reserved                            [10]     IMPLICIT  OctetString,
          ActiveIPAddress                     [11]     IMPLICIT  Unsigned32
}
```

4.1.9.5 EM_ConfiguringDevice service

```
EM_ConfiguringDevice-RequestPDU : : = SEQUENCE {
          DestinationIPAddress                [0]      IMPLICIT  Unsigned32,
          DeviceID                        [1]     IMPLICIT  VisibleString,
          PdTag                           [2]     IMPLICIT  VisibleString,
          AnnunciationInterval            [3]     IMPLICIT  Unsigned16,
          DuplicateTagDetected            [4]     IMPLICIT  Boolean,
          DeviceRedundancyNumber          [5]     IMPLICIT  Unsigned8,
          LANRedunancyPort                [6]     IMPLICIT  Unsigned16,
          DeviceRedundancyState           [7]     IMPLICIT  Unsigned8,
          MaxRedundancyNumber             [8]     IMPLICIT  Unsigned8,
          ActiveIPAddress                 [9]     IMPLICIT  Unsigned32
}
EM_ConfiguringDevice-ResponsePDU : : = CHOICE {
          EM_ConfiguringDevice-PositiveResponsePDU,
          EM_ConfiguringDevice-NegativeResponsePDU
}
EM_ConfiguringDevice-PositiveResponsePDU : : = SEQUENCE {
          DestinationIPAddres         [0]     IMPLICIT  Unsigned32,
          MaxRedundancyNumber         [1]     IMPLICIT  Unsigned8
}
EM_ConfiguringDevice-NegativeResponsePDU : : = SEQUENCE {
          DestinationIPAddress        [0]     IMPLICIT  Unsigned32,
          ErrorType                   [1]     IMPLICIT  ErrorType
}
```

4.1.9.6 EM_SetDefaultValue service

```
EM_SetDefaultValue-RequestPDU : : = SEQUENCE {
        DestinationIPAddress        [0]     IMPLICIT  Unsigned32,
        DeviceID                    [1]     IMPLICIT  VisibleString,
        PdTag                       [2]     IMPLICIT  VisibleString
}
EM_SetDefaultValue-ResponsePDU : : = CHOICE {
        EM_SetDefaultValue-PositiveResponsePDU ,
        EM_SetDefaultValue-NegativeResponsePDU
}
EM_SetDefaultValue-PositiveResponsePDU : : = SEQUENCE {
        DestinationIPAddress        [0]     IMPLICIT  Unsigned32
}
EM_SetDefaultValue-NegativeResponsePDU : : = SEQUENCE {
        DestinationIPAddress        [0]     IMPLICIT  Unsigned32,
        ErrorType                   [1]     IMPLICIT  ErrorType
}
```

4.1.10 Application Access Entity (AAE) services

4.1.10.1 DomainDownload service

```
DomainDownload-RequestPDU : : = SEQUENCE {
        SourceAppID                 [0]   IMPLICIT  Unsigned16,
        DestinationAppID            [1]   IMPLICIT  Unsigned16,
        DestinationObjectID         [2]   IMPLICIT  Unsigned16,
        DataNumber                  [3]   IMPLICIT  Unsigned16,
        MoreFollows                 [4]   IMPLICIT  Boolean,
        Reserved                    [5]   IMPLICIT  OctetString,
        DataLength                  [6]   IMPLICIT  Unsigned16,
        LoadData                    [7]   IMPLICIT  OctetString
}
DomainDownload-ResponsePDU : : = CHOICE {
        DomainDownload-PositiveResponsePDU,
        DomainDownload-NegativeResponsePDU
}
DomainDownload-PositiveResponsePDU : : = SEQUENCE {
        DestinationAppID            [0]   IMPLICIT  Unsigned16
}
DomainDownload-NegativeResponsePDU : : = SEQUENCE {
```

```
          DestinationAppID          [0]     IMPLICIT  Unsigned16,
          Reserved                  [1]     IMPLICIT  OctetString,
          ErrorType                 [2]     IMPLICIT  ErrorType
}
```

4.1.10.2 DomainUpload service

```
DomainUpload-RequestPDU : : = SEQUENCE {
          SourceAppID               [0]   IMPLICIT  Unsigned16,
          DestinationAppID          [1]   IMPLICIT  Unsigned16,
          DestinationObjectID       [2]   IMPLICIT  Unsigned16,
          DataNumber                [3]   IMPLICIT  Unsigned16
}
DomainUpload-ResponsePDU : : = CHOICE {
          DomainUpload-PositiveResponsePDU,
          DomainUpload-NegativeResponsePDU
}
DomainUpload-PositiveResponsePDU : : = SEQUENCE {
          DestinationAppID          [0]   IMPLICIT  Unsigned16,
          DataLength                [1]   IMPLICIT  Unsigned16,
          MoreFollows               [2]   IMPLICIT  Boolean,
          Reserved                  [3]   IMPLICIT  OctetString,
          LoadData                  [4]   IMPLICIT  OctetString
}
DomainUpload-NegativeResponsePDU : : = SEQUENCE {
          DestinationAppID          [0]     IMPLICIT  Unsigned16,
          Reserved                  [1]     IMPLICIT  OctetString,
          ErrorType                 [2]     IMPLICIT  ErrorType
}
```

4.1.10.3 EventRoport service

```
EventRoport-RequestPDU : : = SEQUENCE {
          DestinationAppID          [0]   IMPLICIT  Unsigned16,
          SourceAppID               [1]   IMPLICIT  Unsigned16,
          SourceObjectID            [2]   IMPLICIT  Unsigned16,
          EventNumber               [3]   IMPLICIT  Unsigned16,
          EventData                 [4]   IMPLICIT  OctetString
}
```

4.1.10.4 AcknowledgeEventRoport service

```
AcknowledgeEventRoport-RequestPDU : : = SEQUENCE {
          DestinationAppID          [0]   IMPLICIT  Unsigned16,
          DestinationObjectID       [1]   IMPLICIT  Unsigned16,
          EventNumber               [2]   IMPLICIT  Unsigned16
}
AcknowledgeEventRoport -ResponsePDU : : = CHOICE {
          AcknowledgeEventRoport -PositiveResponsePDU,
          AcknowledgeEventRoport -NegativeResponsePDU
}
AcknowledgeEventRoport -PositiveResponsePDU: : = SEQUENCE{
          DestinationAppID          [0]   IMPLICIT  Unsigned16
}
AcknowledgeEventRoport -NegativeResponsePDU: : = SEQUENCE{
          DestinationAppID          [0]   IMPLICIT  Unsigned16
          Reserved                  [1]   IMPLICIT  OctetString,
          ErrorType                 [2]   IMPLICIT  ErrorType
}
```

4.1.10.5 ReportConditionChanging service

```
ReportConditionChanging-RequestPDU : : = SEQUENCE {
          DestinationAppID          [0]   IMPLICIT  Unsigned16,
          DestinationObjectID       [1]   IMPLICIT  Unsigned16,
          Enabled                   [2]   IMPLICIT  Boolean
}
ReportConditionChanging -ResponsePDU : : = CHOICE {
          ReportConditionChanging -PositiveResponsePDU,
          ReportConditionChanging -NegativeResponsePDU
}
ReportConditionChanging -PositiveResponsePDU: : = SEQUENCE{
          DestinationAppID              [0]   IMPLICIT  Unsigned16
}
ReportConditionChanging -NegativeResponsePDU: : = SEQUENCE{
          DestinationAppID              [0]   IMPLICIT  Unsigned16
          Reserved                      [1]   IMPLICIT  OctetString,
          ErrorType                     [2]   IMPLICIT  ErrorType
}
```

4.1.10.6 Read service

```
Read-RequestPDU : : = SEQUENCE {
          DestinationAppID              [0]   IMPLICIT  Unsigned16,
          DestinationObjectID           [1]   IMPLICIT  Unsigned16,
          SubIndex                      [2]   IMPLICIT  Unsigned16
}
Read -ResponsePDU : : = CHOICE {
          Read -PositiveResponsePDU,
          Read -NegativeResponsePDU
}
Read-PositiveResponsePDU : : = SEQUENCE {
          DestinationAppID              [0]   IMPLICIT  Unsigned16,
          Reserved                      [1]   IMPLICIT  OctetString,
          Data                          [2]   IMPLICIT  OctetString
}
Read-NegativeResponsePDU : : = SEQUENCE {
          DestinationAppID              [0]   IMPLICIT  Unsigned16,
          Reserved                      [1]   IMPLICIT  OctetString,
          ErrorType                     [2]   IMPLICIT  ErrorType
}
```

4.1.10.7 Write service

```
Write-RequestPDU : : = SEQUENCE {
          DestinationAppID              [0]   IMPLICIT  Unsigned16,
          DestinationObjectID           [1]   IMPLICIT  Unsigned16,
          SubIndex                      [2]   IMPLICIT  Unsigned16,
          Reserved                      [3]   IMPLICIT  OctetString,
          Data                          [4]   IMPLICIT  OctetString
}
Write -ResponsePDU : : = CHOICE {
          Write -PositiveResponsePDU,
          Write -NegativeResponsePDU
}
Write -PositiveResponsePDU : : = SEQUENCE {
          DestinationAppID              [0]   IMPLICIT  Unsigned16,
}
Write -NegativeResponsePDU : : = SEQUENCE {
          DestinationAppID              [0]   IMPLICIT  Unsigned16,
          Reserved                      [1]   IMPLICIT  OctetString,
          ErrorType                     [2]   IMPLICIT  ErrorType
}
```

4.1.10.8 VariableDistribute service

```
VariableDistribute-RequestPDU : : = SEQUENCE {
          SourceAppID                   [0]   IMPLICIT  Unsigned16,
          SourceObjectID                [1]   IMPLICIT  Unsigned16,
          Data                          [2]   IMPLICIT  OctetString
}
```

4.1.10.9 FRTRead service

```
FRTRead-RequestPDU : : = SEQUENCE {
          DestinationObjectID          [0]   IMPLICIT  Unsigned16,
          SubIndex                     [1]   IMPLICIT  Unsigned16
}
FRTRead -ResponsePDU : : = CHOICE {
          FRTRead -PositiveResponsePDU,
          FRTRead -NegativeResponsePDU
}
FRTRead-PositiveResponsePDU : : = SEQUENCE {
          FRTData                      [0]   IMPLICIT  OctetString
}
FRTRead-NegativeResponsePDU : : = SEQUENCE {
          ErrorType                    [0]   IMPLICIT  ErrorType
}
```

4.1.10.10 FRTWrite service

```
FRTWrite-RequestPDU : : = SEQUENCE {
          DestinationObjectID          [0]   IMPLICIT  Unsigned16,
          SubIndex                     [1]   IMPLICIT  Unsigned16,
          Reserved                     [2]   IMPLICIT  OctetString,
          Data                         [3]   IMPLICIT  OctetString
}
FRTWrite -ResponsePDU : : = CHOICE {
          FRTWrite -PositiveResponsePDU,
          FRTWrite -NegativeResponsePDU
}
FRTWrite -PositiveResponsePDU : : = SEQUENCE {
          DestinationObjectID          [0]   IMPLICIT  Unsigned16,
}
FRTWrite -NegativeResponsePDU : : = SEQUENCE {
          ErrorType                    [0]   IMPLICIT  ErrorType
}
```

4.1.10.11 FRTVariableDistribute service

```
FRTVariableDistribute-RequestPDU : : = SEQUENCE {
          SourceObjectID               [0]   IMPLICIT  Unsigned16,
          Data                         [1]   IMPLICIT  OctetString
}
```

4.1.10.12 BlockTransmissionOpen service

```
BlockTransmissionOpen-RequestPDU : : = SEQUENCE {
          SourceAppID                  [0]   IMPLICIT  Unsigned16,
          DestinationAppID             [1]   IMPLICIT  Unsigned16
          DestinationObjectID          [2]   IMPLICIT  Unsigned16
          BlockType                    [3]   IMPLICIT  Unsigned16
          BlockConfigInfo              [4]   IMPLICIT  OctetString
}
BlockTransmissionOpen-ResponsePDU : : = CHOICE {
          BlockTransmissionOpen -PositiveResponsePDU,
          BlockTransmissionOpen -NegativeResponsePDU
}
BlockTransmissionOpen-PositiveResponsePDU : : = SEQUENCE {
          DestinationAppID             [0]   IMPLICIT  Unsigned16,
          Reserved                     [1]   IMPLICIT  OctetString,
          MultiIPAddress               [2]   IMPLICIT  Unsigned32
}
BlockTransmissionOpen-NegativeResponsePDU : : = SEQUENCE {
          DestinationAppID             [0]   IMPLICIT  Unsigned16,
          Reserved                     [1]   IMPLICIT  OctetString,
          ErrorType                    [2]   IMPLICIT  ErrorType
}
```

4.1.10.13 BlockTransmissionClose service

```
BlockTransmissionClose-RequestPDU : : = SEQUENCE {
          SourceAppID                  [0]   IMPLICIT  Unsigned16,
          DestinationAppID             [1]   IMPLICIT  Unsigned16
          DestinationObjectID          [2]   IMPLICIT  Unsigned16
          BlockType                    [3]   IMPLICIT  Unsigned16
}
BlockTransmissionClose-ResponsePDU : : = CHOICE {
          BlockTransmissionClose -PositiveResponsePDU,
          BlockTransmissionClose -NegativeResponsePDU
}
BlockTransmissionClose-PositiveResponsePDU : : = SEQUENCE {
          DestinationAppID             [0]   IMPLICIT  Unsigned16,
}
BlockTransmissionClose-NegativeResponsePDU : : = SEQUENCE {
          DestinationAppID             [0]   IMPLICIT  Unsigned16,
          Reserved                     [1]   IMPLICIT  OctetString,
          ErrorType                    [2]   IMPLICIT  ErrorType
}
```

4.1.10.14 BlockTransmit service

```
BlockTransmit-RequestPDU : : = SEQUENCE {
          SourceAppID                  [0]   IMPLICIT  Unsigned16,
          DestinationAppID             [1]   IMPLICIT  Unsigned16,
          DestinationObjectID          [2]   IMPLICIT  Unsigned16,
          DataLength                   [3]   IMPLICIT  Unsigned16,
          BlockType                    [4]   IMPLICIT  Unsigned16
          SequenceNumber               [5]   IMPLICIT  Unsigned16
          TimeStamp                    [6]   IMPLICIT  PrecisionTimeDifference
          SendCount                    [7]   IMPLICIT  Unsigned16
          BlockData                    [8]   IMPLICIT  OctetString
}
```

4.1.10.15 BlockTransmissionHeartbeat service

```
BlockTransmissionHeartbeat-RequestPDU : : = SEQUENCE {
          SourceAppID                  [0]  IMPLICIT  Unsigned16,
          DestinationAppID             [1]   IMPLICIT  Unsigned16,
          DestinationObjectID          [2]   IMPLICIT  Unsigned16,
          ReceptionCount               [3]   IMPLICIT  Unsigned16,
          CumulativeLost               [4]   IMPLICIT  Unsigned16
          Jitter                       [5]   IMPLICIT  PrecisionTimeDifference
}
```

4.1.11 Abstract syntax of data type

4.1.11.1 Notation of Boolean type

```
          Boolean ::= BOOLEAN                           --value is non-zero means TRUE
                                                        --value is zero means FALSE
```

4.1.11.2 Notation of integer type

```
          Int8 ::= INTEGER  (-128..+127)                -- integer range -27<=  i  <=  27-1
          Int16 ::= INTEGER (-32 768..+32 767)          -- integer range -215<=  i  <=  215-1
          Int32 ::= INTEGER                             -- integer range -231<=  i  <=  231-1
          Int64 ::= INTEGER                             -- integer range -263<=  i  <=  263-1
```

4.1.11.3 Notation of unsigned integer type

```
          Unsigned8 ::= INTEGER  (0..255)               -- integer range 0 <=  i  <=  28-1
          Unsigned16 ::= INTEGER (0..65 535)            -- integer range 0 <=  i  <=  216-1
          Unsigned32 ::= INTEGER                        -- integer range 0 <=  i  <=  232-1
          Unsigned64 ::= INTEGER                        -- integer range 0 <=  i  <=  264-1
```

4.1.11.4 Notation of float data type

```
Real ::= BIT STRING  SIZE (4)          -- IEC-60559 single precision
```

4.1.11.5 Notation of visible string type

```
VisibleString ::= VISIBLE STRING       --general use
```

4.1.11.6 Notation of octet string type

```
OctetString ::= Octet STRING           --general use
```

4.1.11.7 Notation of bit string type

```
BitString ::= BIT STRING               -- general use
```

4.1.11.8 Notation of TimeofDay type

```
TimeOfDay ::= OctetString6
```

4.1.11.9 Notation of binary date type

```
BinaryDate ::= OctetString8
```

4.2 Object definitions in FAL management ASE

4.2.1 Type 14 MOB Header

The object of the Type 14 MOB Header is defined as shown in Table 2.

Table 2 – Definition of Type 14 MOB header object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of Type 14 MOB header object in the Type 14 MOB
2	MOB Revision Number	Read Only	Unsigned16	2	2	The version of Type 14 MOB

4.2.2 Type 14 device descriptor object

The object of the Type 14 Device Descriptor Object is defined as shown in Table 3.

Table 3 – Definition of Type 14 device descriptor object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of Type 14 Device Descriptor Object in the MOB
2	Reserved	Read Only	Unsigned8	2	1	reserved
3	Application Type	Read Only	Unsigned8	3	1	application type
4	Device ID	Read Only	VisibleString	4	32	device ID
5	PD_Tag	Read Only	VisibleString	36	32	device Tag
6	Active IP Address	Read Only	Unsigned32	68	4	current operational IP address
7	Device Type	Read Only	Unsigned8	72	1	device type
8	Status	Read Only	Unsigned8	73	1	device status
9	Device Version	Read Only	Unsigned16	74	2	device version number
10	Annunciation Interval	Read Only	Unsigned16	76	2	the interval of devices broadcast its annunciation
11	Annunciation Version Number	Read Only	Unsigned16	78	2	annunciation version number of devices broadcast

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
12	Device Redundancy State	Read Only	Unsigned8	80	1	device redundancy status
13	Device Redundancy Number	Read Only	Unsigned8	81	1	device redundancy number
14	LANRedundancyPort	Read Only	Unsigned16	82	2	redundant messages processing port of the device
15	Max Redundancy Number	Read Only	Unsigned8	84	1	maximum redundancy number of the device
16	Duplicate Tag Detected	Read Only	Boolean	85	1	this property describes whether the device's PD_Tag is in collision with another device

4.2.3 Time synchronization object

The object of the Time Synchronization Object class is defined as shown in Table 4:

Table 4 – Definition of the time synchronization object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of the Time Synchronization Object in the MOB
2	Reserved	Read Only	OctetString	2	2	reserved
3	Primary Time Server	Read/Write	Unsigned32	4	4	IP address of master time server
4	Secondary Time Server	Read/Write	Unsigned32	8	4	IP address of slave time server
5	Time Request Timeout	Read Only	Unsigned32	12	4	the maximum time that time client waits for the response of time server in seconds
6	Time Request Interval	Read/Write	Unsigned32	16	4	the time interval that time client requests the time server
7	Capable Time Sync Class	Read Only	Unsigned32	20	4	the synchronization precision supported by time client
8	Target Time Sync Class	Read/Write	Unsigned32	24	4	the required synchronization precision as to the time client
9	Current Time	Read Only	BinaryDate	28	8	current time of the device
10	Standard Time Difference	Read Only	PrecisionTimeDifference	36	8	Standard time difference

4.2.4 Maximum response time object

The object of the Maximum Response Time Object class is defined as shown in Table 5.

Table 5 – Definition of maximum response time object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of object in the MOB
2	Reserved	Read Only	OctetString	2	2	reserved

3	Max Response Time	Read/Write	PrecisionTimeDifference	4	8	the maximum response time of confirmed service in nanoseconds

4.2.5 Type 14 communication scheduling management object

The object of the Type 14 Communication Scheduling Management class is defined as shown in Table 6.

Table 6 – Definition of the Type 14 communication scheduling management object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of object in the MOB
2	Reserved	Read Only	OctetString	2	2	reserved
3	Communication MacroCycle	Read/Write	PrecisionTimeDifference	4	8	the communication macro period of subnet which the device belongs to. The unit is nanoseconds
4	NonPeriodic Data Transfer Offset	Read/Write	PrecisionTimeDifference	12	8	the offset of non-periodic message begin to transmit relative to the start of communication macro period. The unit is nanoseconds
5	Communication Macrocycle Version Number	Read Only	Unsigned16	20	2	the version number of communication macro period

4.2.6 Device application information object

The object of the Device Application Information class is defined as shown in Table 7.

Table 7 – Definition of the device application information object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of object in the MOB
2	XDDL Version	Read Only	Unsigned16	2	2	the device description version number

4.2.7 FB application information header

The object of the Encoding of FB Application Information Header class is defined as shown in Table 8.

Table 8 – Definition of FB application information header

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of object in the MOB
2	Number of FB Application Information Object	Read Only	Unsigned16	2	2	the number of FB Application Information Object
3	First Number of FB Application Information Object	Read Only	Unsigned16	4	2	first Number of FB Application Information Object

4.2.8 Domain application information header

The object of the Domain Application Information Header class is defined as shown in Table 9.

Table 9 – Definition of domain application information header

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of Domain Application Information Header object in the MOB
2	Number of Domain Application Information Object	Read Only	Unsigned16	2	2	number of Domain Application Information Objects in the device
3	First Number of Domain Application Object	Read Only	Unsigned16	4	2	first Number of Domain Application Object in the MOB
4	Number of Configured Domain Object	Read Only	Unsigned16	6	2	number of Configured Domain Objects
5	Number of UnConfigured Domain Object	Read Only	Unsigned16	8	2	number of UnConfigured Domain Objects

4.2.9 Type 14 link object header

The object of the Type 14 Link Object Header class is defined as shown in Table 10.

Table 10 – Definition of Type 14 link object header

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of Link Object Header t in the MOB
2	Number of Link Object	Read Only	Unsigned16	2	2	number of Link Object in the device
3	First Number of Link Object	Read Only	Unsigned16	4	2	first Number of Link Object in the MOB
4	Number of Configured Link Object	Read Only	Unsigned16	6	2	number of Configured Link Objects
5	Number of UnConfigured Link Object	Read Only	Unsigned16	8	2	number of UnConfigured Link Objects

4.2.10 Type 14 link object header

The object of the Type 14 Link Object Header class is defined as shown in Table 11.

Table 11 – Definition of Type 14 FRT link object header

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of FRT Link Object Header t in the MOB
2	Number of FRT Link Object	Read Only	Unsigned16	2	2	number of FRT Link Object in the device
3	First Number of FRT Link Object	Read Only	Unsigned16	4	2	first Number of FRT Link Object in the MOB
4	Number of Configured FRT Link Object	Read Only	Unsigned16	6	2	number of Configured FRT Link Objects
5	Number of UnConfigured FRT Link Object	Read Only	Unsigned16	8	2	number of UnConfigured FRT Link Objects

4.2.11 FB application information object

The object of the FB Application Information class is defined as shown in Table 12.

Table 12 – Definition of FB application information object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of FB Application Information Object in the MOB
2	Reserved	Read Only	Unsigned16	2	2	reserved
3	FB Name	Read Only	VisibleString	4	32	FB Name, its length is 32 octets, if the string length is less than 32 octets, then the remained part are padded with BLANK(0x20)
4	FB Type	Read Only	Unsigned16	36	2	FB Type
5	Max Number of Instantiation	Read Only	Unsigned16	38	2	the maximum instances number of FB
6	FB Execution Time	Read Only	Unsigned32	40	4	the execution time of FB in milliseconds
7	First Number of Instantiation	Read Only	Unsigned16	44	2	first Instance Number allocated to FB instance when it is instantiated

4.2.12 Type 14 link object

The object of the Type 14 Link Object class is defined as shown in Table 13.

Table 13 – Definition of Type 14 link object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of Type 14 Link Object in the MOB
2	LocalAppID	Read/Write	Unsigned16	2	2	instance ID of local instance
3	Local Object ID	Read/Write	Unsigned16	4	2	the index of local variable object
4	RemoteAppID	Read/Write	Unsigned16	6	2	instance ID of remote FB
5	RemoteObjectID	Read/Write	Unsigned16	8	2	ID of remote element object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
6	ServiceOperation	Read/Write	Unsigned8	10	1	Type 14 service ID used by Link Object
7	ServiceRole	Read/Write	Unsigned8	11	1	role of local object in the communication process
8	RemoteIPAddress	Read/Write	Unsigned32	12	4	IP address of remote device; if local and destination FB instance objects are in the same Type 14 device, then this property can be ignored; if the Type 14 service use the broadcast or multicast method, then this property should be broadcast or multicast group address
9	SendTimeOffset	Read/Write	PrecisionTime Difference	16	8	time offset when sending periodic packet from the start time of a communication macrocycle. Its data type is 4 octets of TimeDifference. The unit is nanoseconds

4.2.13 Type 14 FRT link object

The object of the Type 14 FRT Link Object class is defined as shown in Table 14.

Table 14 – Definition of Type 14 FRT link object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of Type 14 Link Object in the MOB
2	Local Object ID	Read/Write	Unsigned16	2	2	the index of local variable object
3	RemoteObjectID	Read/Write	Unsigned16	4	2	ID of remote element object
4	ServiceOperation	Read/Write	Unsigned8	6	1	Type 14 service ID used by Link Object
5	ServiceRole	Read/Write	Unsigned8	7	1	role of local object in the communication process
6	RemoteMACAddress	Read/Write	Unsigned32	8	4	MAC address of remote device; if local and destination FB instance objects are in the same Type 14 device, then this property can be ignored; if the Type 14 service uses the broadcast or multicast method, then this property should be broadcast or multicast group address
7	SendTimeOffset	Read/Write	PrecisionTime Difference	12	8	time offset when sending periodic packet from the start time of a communication macrocycle. Its data type is 4 octets of TimeDifference. The unit is nanoseconds
8	ValidBitOffset	Read/Write	Unsigned16	20	4	the bit offset when the relevant message should be sent or received from the start time of field of DATA in FRTVariableDistribute Service
9	ValidBitNumber	Read/Write	Unsigned16	24	4	the bit number when the relevant message should be sent or received from the start time of field of DATA in FRTVariableDistribute Service

4.2.14 Domain application information object

The object of the Domain Application Information class is defined as shown in Table 15.

Table 15 – Definition of domain application information object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length	Description
1	Object ID	Read Only	Unsigned16	0	2	the index of the domain application information object in the MOB
2	Domain Object ID	Read Only	Unsigned16	2	2	the index of the domain object corresponding to the domain application information object
3	ConfigurationStatus	Read Only	Boolen	4	1	the configuration status of domain object, Boolean type, if its value is TRUE, then it shows that the domain object is not configured
4	Reserved	Read Only	OctetString	5	3	reserved
5	Domain Name	Read Only	Unsigned16	8	32	the name of the domain object, its length is 32 octets, the unused part is padded with BLANK (0x20)

4.3 Definition of objects used in Type 14 application access entity

Subclause 4.3 defines the encodings of objects in Type 14 application access entity.

4.3.1 Domain object

The object of the Domain class is defined as shown in Table 16.

Table 16 – Definition of domain object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
1	ObjectID	Read Only	Unsigned16	2	the index of Domain Object
2	Domain Name	Read/Write	VisibleString	32	the name of Domain Object
3	Max Octets	Read Only	Unsigned16	2	the maximum octets in the domain
4	Password	Read/Write	Unsigned16	2	the password used to access the Domain Object
5	AccessGroups	Read/Write	Unsigned8	1	the access group of Domain Object
6	AccessRights	Read/Write	Unsigned8	1	the access right of Domain Object
7	Local Address	Read Only	Unsigned32	4	the pointer pointed to the specific Domain Object. If not used, its value should be set to 0xFFFF FFFF
8	Domain State	Read Only	Unsigned8	1	the status of domain object, it can be the following value: 0: EXISTENT 1: DOWNLOADING 2: UPLOADING

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
					3: READY 4: IN-USE
9	Last State	Read Only	Unsigned8	1	the status of domain object before upload/download, the meaning of its value if shown as follows: 0: EXISTENT 1: DOWNLOADING 2: UPLOADING 3: READY 4: IN-USE
10	Used Application Counter	Read Only	Unsigned16	2	the number of programs using the domain now, if the counter value is bigger than 0, it shows that this domain is being used, so it cannot be overwritten by the download service, its data type is unsigned16

4.3.2 Simple variable object

The definition of Simple Variable Object is shown in Table 17.

Table 17 – Definition of simple variable object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
1	ObjectID	Read Only	Unsigned16	2	the index of the Variable Object in the MOB
2	Data Type	Read Only	Unsigned8	1	the data type of the Variable Object
3	Length	Read Only	Unsigned16	2	the length of Variable Object in octet
4	Local Address	Read Only	Unsigned32	4	the pointer pointed to the specific Variable Object which can be used to internally address the Domain Object. If not used , its value should be set to 0xFFFF FFFF
5	Password	Read/Write	Unsigned16	2	the password used to access the Variable Object
6	AccessGroups	Read/Write	Unsigned8	1	the access group of Variable Object
7	AccessRights	Read/Write	Unsigned8	1	the access right to Variable Object

4.3.3 Event object

The definition of Event Object is shown in Table 18.

Table 18 – Definition of event object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
1	ObjectID	Read Only	Unsigned16	2	the ID of the Event Object
2	Length	Read Only	Unsigned16	2	the octet length of the Event Object
3	Password	Read/Write	Unsigned16	2	the password used to access the Domain Object
4	AccessGroups	Read/Write	Unsigned8	1	the access group of the Domain Object
5	AccessRights	Read/Write	Unsigned8	1	the access right of the Domain Object
6	Local Address	Read Only	Unsigned32	4	the pointer pointed to the specific Event

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
					Object, it is used to internally address the Variable Object. If not used , its value should be set to 0xFFFF FFFF
7	Enabled	Read Only	Boolean	1	Enabled = TRUE ⇔ UNLOCKED Signifies the event object isn't locked, and the event can be sent out Enabled = FALSE ⇔ LOCKED Signifies the event object is locked, and the event cannot be sent out

4.3.4 Type 14 socket mapping object

The definition of Type 14 Socket Mapping Object is shown in Table 19.

Table 19 – Definition of Type 14 socket mapping object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
1	LocalIPAddress	Read Only	Unsigned32	4	IP address of local device
2	RemoteIPAddress	Read Only	Unsigned32	4	IP address of remote device
3	ActiveUdpPort	Read Only	Unsigned16	2	the UDP port used when sending message
4	ActiveServiceID	Read Only	Unsigned16	2	Service ID
5	ActiveMesssagLength	Read Only	Unsigned16	2	the length of the message waiting to be sent
6	ActiveMessageID	Read Only	Unsigned16	2	Active packet ID, i.e. the MessageID
7	ActiveMesssageTime	Read Only	Time Difference	6	the response time of the active message, this parameter shows the maximum response time of the active message, if it does not need the response, it should be set to zero
8	ActiveDataPointer	Read Only	Unsigned32	4	the pointer to the header of the active message
9	MaxMessageLength	Read Only	Unsigned16	2	the maximum message length allowed. If the user level message exceeds the length, it will be denied to send, and will return an error flag
10	MaxRetransmitNumber	Read Only	Unsigned16	2	the maximum retransmission times allowed

4.3.5 Type 14 socket timer object

The definition of the Type 14 Socket Timer Object is shown in Table 20.

Table 20 – Definition of Type 14 socket timer object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
1	TimerID	Read Only	Unsigned16	2	the ID of the timer
2	ActiveServiceID	Read Only	Unsigned16	2	Service ID which indicates the service used to send the message
3	ActiveMessageID	Read Only	Unsigned16	2	the active message ID, i.e. Message ID in the message

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
4	ActiveMessageTime	Read Only	Unsigned32	4	the response time of the active message, this parameter shows the maximum response time of the active message, if it does not need the response, it shall be set to zero. The unit is millisecond. The unit is milliseconds for RT applications, and microseconds for FRT applications

4.3.6 ErrorType object

The definition of the ErrorType Object is shown in Table 21.

Table 21 – Definition of ErrorType object

No.	Parameter name	Read/write property	Data type	Octet offset	Octet length
1	Error Class	Read Only	Int8	1	Error class
2	Error Code	Read Only	Int8	1	Error code, this parameter gives a more concrete error description
3	Additional Code	Read Only	Int8	1	Additional code, this parameter is optional, and the user can define its usage according to its own need
4	Reserved	Read Only	Octetstring	1	reserved
5	Additional Description	Read Only	VisibleString	32	Additional description, this parameter is optional, and it used to add a text description in the error information. Its data type is VisibleString

5 Transfer syntax

5.1 Encoding of basic data types

5.1.1 Boolean

A Boolean value is encoded in one octet, all bits are set to 0 for FALSE and 1 for TRUE.

If the value is TRUE, the value of each bit is shown in Table 22 (Bit 7 is MSB, Bit 0 is LSB).

Table 22 – Encoding of Boolean value TRUE

Octet	Bit							
	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1

If the value is FALSE, the value of each bit is shown in Table 23.

Table 23 – Encoding of Boolean value FALSE

Octet	Bit							
	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0

5.1.2 Unsigned8

The Unsigned8 type is encoded in one octet, the range is from 0 to 255, the weight of each bit is shown in Table 24.

Table 24 – Encoding of Unsigned8 data type

Octet	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

5.1.3 Unsigned16

The Unsigned16 type is encoded in two octets, the range is from 0 to 216-1, the weight of each bit is shown in Table 25.

Table 25 – Encoding of Unsigned16 data type

Octet	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

5.1.4 Unsigned32

The Unsigned32 type is encoded in four octets, the range is from 0 to 232-1, the weight of each bit is shown in Table 26.

Table 26 – Encoding of Unsigned32 data type

Octet	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8
4	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

5.1.5 Unsigned64

The Unsigned64 type is encoded in eight octets, the range is from 0 to 264-1, the weight of each bit is shown in Table 27.

Table 27 – Encoding of Unsigned64 data type

Octet	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{63}	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}
2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}

Octet	Bit							
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.6 Int8

The Int8 type is encoded in one octet, the range is from -128 to 127. If the sign bit (SN) is 1, the data is negative, otherwise, the data is positive or zero when bit SN is 0. The weight of each bit is shown in Table 28.

Table 28 – Encoding of Int8 data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	SN	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.7 Int16

The Int16 type is encoded in two octets, the range is from -2^{15} to 2^{15}-1. If bit SN is 1, the data is negative; otherwise, the data is positive or zero when bit SN is 0. The weight of each bit is shown in Table 29.

Table 29 – Encoding of Int16 data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	SN	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.8 Int32

The Int32 type is encoded in four octets, the range is from -2^{31} to 2^{31}-1. If the SN is 1, the data is negative; otherwise, the data is positive or zero when bit SN is 0. The weight of each bit is shown in Table 30.

Table 30 – Encoding of Int32 data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	SN	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.9 Int64

The Int64 type is encoded in eight octets, the range is from -2^{63} to 2^{63}-1. If the value of bit SN is 1, the data is negative; otherwise, the data is positive or zero when bit SN is 0. The weight of each bit is shown in Table 31.

Table 31 – Encoding of Int64 data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	SN	2^{62}	2^{61}	2^{60}	2^{59}	2^{58}	2^{57}	2^{56}
2	2^{55}	2^{54}	2^{53}	2^{52}	2^{51}	2^{50}	2^{49}	2^{48}
3	2^{47}	2^{46}	2^{45}	2^{44}	2^{43}	2^{42}	2^{41}	2^{40}
4	2^{39}	2^{38}	2^{37}	2^{36}	2^{35}	2^{34}	2^{33}	2^{32}
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.10 Real

The Real type is encoded in four octets. The range shall be according to IEEE Std 754 Short Real Number (total 32 bits). If the value of bit SN is 1, the data is negative, otherwise, the data is positive or zero when bit SN is 0. Bits 6 to 0 of octet 1 and bit 7 of octet 2 defines the exponent field. It is followed by the fraction field from bit 6 of octet 1 to bit 0 of octet 2. The weight of each bit is shown in Table 32.

Table 32 – Encoding of Real type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	SN	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}
2	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
3	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
4	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}

5.1.11 VisibleString

The VisibleString type is encoded in visible string, the length is variable. The definition shall be according to ISO/IEC 646 and ISO/IEC 2375. The encoding is shown in Table 33.

Table 33 – Encoding of VisibleString data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	the first character							
2	the second character							
……	and so on……							
……	so on……							
N								

5.1.12 OctetString

The OctetString type is encoded in one or several octets which are aligned from 1 to n according to the sequence number. The encoding is shown in Table 34.

Table 34 – Encoding of OctetString data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	Binary data							
2	Binary data							
……	……							
……	……							
N								

5.1.13 BitString

The BitString type is encoded in a group of Octets, the length is variable from 1 to n, n is an arbitrary natural number. The encoding is shown in Table 35.

Table 35 – Encoding of BitString data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
……	so on……							
……	so on……							
n								

5.1.14 TimeOfDay

The TimeOfDay type consists of a date and a time, it is encoded in total 6 octets. The date field is encoded as Unsigned16 data (octet 1 and octet 2), it is stated in days relatively to the first of January 2000. On the first of January 2000, the date starts with the value zero. The time is stated in milliseconds since midnight, at midnight the counting starts with the value zero. The time field is encoded as an Unsigned32 data (from octet 3 to octet 6). The encoding is shown in Table 36.

```
{
        Unsigned16  Date;           //days
        Unsigned32  Millisecond;    //milliseconds
}
```

Table 36 – Encoding of TimeOfDay data type

Octet	Bit							
	7	6	5	4	3	2	1	0
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
3	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
4	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.15 BinaryDate

The type BinaryDate consists of a calendar date and a time, it is encoded in 8 octets shown in Table 37.

The year field is encoded as an Unsigned16 data (octet 1 and octet 2), if its value is 2004, it represents the year of 2004.

The month field is encoded as an Unsigned8 data (octet 3), the range is from 1 to 12, its value represents one of 12 months in a year.

The date field is encoded as an Unsigned8 data (octet 4), the range is from 1 to 31, its value represents one day of 31 days in a month.

The hour field is encoded as an Unsigned8 data (octet 5), the range from 0 to 23, its value represents 1 h of 24 h in a day.

The minute field is encoded as an Unsigned8 data (octet 6), the range is from 0 to 59, its value represents 1 min of 60 min in an hour.

The millisecond field is encoded as an Unsigned16 data (octet 7 and octet 8), the range is from 0 to 59999, its value represents 1 ms of 60 000 ms in a minute.

```
{
    Unsigned16  Year;               //year
    Unsigned8   Month;              //month
    Unsigned8   Date;               //day
    Unsigned8   Hour;               //hour
    Unsigned8   Minute;             //minute
    Unsigned16   Millisecond;       //millisecond
}
```

Table 37 – Encoding of BinaryDate data type

Octet	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
2	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
3	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
5	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.16 PrecisionTimeDifference

The type PrecisionTimeDifference consists of second and nanosecond, it is encoded in 8 octets ; its value states the time difference.

The Second field is encoded as an Unsigned32 data (form octet 1 to octet 4), its value represents the seconds difference.

The Nanosecond field is stated in macroseconds and encoded as an Int32 data (from octet 5 to octet 8), its value represents the macroseconds difference. The sign of Nanosecond belongs to the data type. The encoding is shown in Table 38.

```
{
    Unsigned32  Second;              //second difference
    Int32  Nanosecond;               // Nanosecond with sign
}
```

Table 38 – Encoding of PrecisionTimeDifference data type

Octet	Bit							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
5	SN	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}
6	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
7	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}
8	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}

5.1.17 Encoding of SEQUENCE

The SEQUENCE structure is comparable with a record. It represents a collection of user data of the same or of different Data Types.

A structure may contain a simple variable or further structures as components.

5.1.18 Encoding of CHOICE

A CHOICE represents a selection from a set of predefined possibilities.

5.2 Encoding of Type 14 APDU header

The encoding of the Type 14 Application layer Service Message packet header is shown in Table 39.

Table 39 – Encoding of Type 14 application layer service message header

No.	Parameter Name	Data type	Octet offset	Octet length	Description
1	ServiceID	Unsigned8	0	1	this parameter describes the service type and message type. Bit 7 to 6 indicates the message type: 00: request message 01: response message 10: error message 11: reserved The lowest six bits used to signify the service ID
2	Reserved	OctetString	1	3	reserved
3	Length	Unsigned16	4	2	this parameter describes the length of

No.	Parameter Name	Data type	Octet offset	Octet length	Description
					the whole message
4	MessageID	Unsigned16	6	2	this parameter describes the ID of the message

5.3 Encoding of FAL management entity service parameters

5.3.1 EM_DetectingDevice service

The EM_DetectingDevice request parameters are coded as shown in Table 40.

Table 40 – Encoding of EM_DetectingDevice request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Query Type	Unsigned8	0	1	signifies the query request type: 0: Query according to PD_Tag. The following parameter is PD_Tag 1: Query according to FB Tag. The following parameter is FB Tag 2: Query according to ElementID. The following parameter is FB Tag and ElementID
2	Reserved	Octetstring	1	3	reserved
3	PD_Tag	VisibleString	4	32	the physical device tag, its length is 32 octets, and the unused part is padded with BLANK (0x20)
4	FB Tag	VisibleString	36	32	function block instance tag which is used to query the information of Type 14 device which include the FB instance
5	Element ID	Unsigned16	68	2	Element ID in FB which must be used with FB Tag together, because ElementID is unique only in one FB instance

5.3.2 EM_OnlineReply service

The EM_OnlineReply request parameters are coded as shown in Table 41.

Table 41 – Encoding of EM_OnlineReply request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Query Type	Unsigned8	0	1	signifies the Query Type: 0: Query according to PD_Tag 1: Query according to FB Tag 2: Query according to ElementID
2	Duplicate Tag Detected	Boolean	1	1	this property describes if the device’s PD_Tag collides with the other devices’ PD_Tag (i.e. repeated PD_Tag). TRUE= PD_Tags Collide
3	Reserved	Octetstring	2	2	reserved
4	Queried Object IP Address	Unsigned32	4	4	IP address of the queried Type 14 physical device (i.e. the IP address of the local device)
5	Queried Object Device ID	VisibleString	8	32	the queried Type 14 physical device ID (e.g. local device ID), its length is 32 octets, and the unused part is padded with BLANK (0x20)
6	Queried	VisibleString	40	32	the queried Type 14 physical device tag (e.g.

No.	Parameter name	Data type	Octet offset	Octet length	Description
	Object PD_Tag				local device tag). Its length is 32 octets, and the unused part is padded with BLANK (0x20)

5.3.3 EM_GetDeviceAttribute service

5.3.3.1 Request primitive

The EM_GetDeviceAttribute request parameters are coded as shown in Table 42.

Table 42 – Encoding of EM_GetDeviceAttribute request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Destination IP Addres	Unsigned32	0	4	IP address of the target device

5.3.3.2 Positive response primitive

The EM_GetDeviceAttribute positive response parameters are coded as shown in Table 43.

Table 43 – Encoding of EM_GetDeviceAttribute positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Device ID	VisibleString	0	32	Local device ID
2	PD_Tag	VisibleString	32	32	the physical device tag, its length is 32 octets, and the unused part is padded with BLANK (0x20)
3	Status	Unsigned8	64	1	the status of the Type 14 device: 0: no address 1: unconfigured 2: configured, operational
4	Device Type	Unsigned8	65	1	local device type
5	Annunciation Interval	Unsigned16	66	2	the interval of the annunciation message sent out by the device
6	Annunciation Version Number	Unsigned16	68	2	the version number of the annunciation message
7	Duplicate Tag Detected	Boolean	70	1	this property indicates whether PD_Tag of the device is in collision with the other or not (i.e. duplicated PD_Tag). TRUE= PD_Tag in collision
8	Redundancy Number	Unsigned8	71	1	the redundancy number of local device, if the device is active, then this value is zero, and has no following parameters
9	Device Redundancy State	Unsigned8	72	1	the redundancy status the local device is in: 0: active status 1: back-up status If the redundancy number is 0, then the response primitive does not contain this parameter

No.	Parameter name	Data type	Octet offset	Octet length	Description
10	Max Redundancy Number	Unsigned8	73	1	the maximum redundancy number of the device, if the redundancy number is 0, then the response primitive does not contain this parameter
11	Reserved	Octetstring	74	2	reserved
12	Active IP Address	Unsigned32	76	4	IP address of the active device (if no redundancy, then its local IP address); if the redundancy number is 0, then the response primitive does not contain this parameter

5.3.3.3 Negative response primitive

The EM_GetDeviceAttribute negative response parameters are coded as shown in Table 44.

Table 44 – Encoding of EM_GetDeviceAttribute negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationIPAddress	Unsigned32	0	4	IP address of the target device
2	Error Type	ErrorType	4	N	see ErrorType

5.3.4 EM_ActiveNotification service

The EM_ActiveNotification request parameters are coded as shown in Table 45.

Table 45 – Encoding of EM_ActiveNotification request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Device ID	VisibleString	0	32	ID of local device
2	PD_Tag	VisibleString	32	32	the local physical device tag, its length is 32 octets, and the unused part is padded with BLANK (0x20)

3	Status	Unsigned8	64	1	the status of the Type 14 device:
					0: no address;
					1: unconfigured;
					2: configured,operational

4	Device Type	Unsigned8	65	1	local device type
5	Annunciation version number	Unsigned16	66	2	the version number of the annunciation message
6	Device Redundancy Number	Unsigned8	68	1	the redundancy number of local device, if it is an active device, then this value is 0, otherwise the following parameter is invalid
7	Device Redundancy State	Unsigned8	69	1	the redundancy status of the local device: 0: active state 1: backup state If the redundancy number is 0, then the response primitive does not contain this parameter
8	LAN Redundancy Port	Unsigned16	70	2	the LAN redundancy message processing port of the device which requests the service
9	Duplicate Tag Detected	Boolean	72	1	this property describes if the device's PD_Tag collides with the other devices' PD_Tag (i.e. duplicated PD_Tag). TRUE= PD_Tags Collide
10	Reserved	Octetstring	73	2	reserved
11	Max. Redundancy Number	Unsigned8	75	1	the maximum redundancy number of the device
12	Active IP Address	Unsigned32	76	4	IP address of the active device (if no redundancy, then its local IP address); if the redundancy number is 0, then the response primitive does not contain this parameter

5.3.5 EM_ConfiguringDevice service

5.3.5.1 Request Primitive

The EM_ConfiguringDevice request parameters are coded as shown in Table 46.

Table 46 – Encoding of EM_ConfiguringDevice request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationIPAddress	Unsigned32	0	4	Destination IP Address
2	Device ID	VisibleString	4	32	ID of local device
3	PD_Tag	VisibleString	36	32	the local physical device tag, its length is 32 octets, and the unused part is padded with BLANK (0x20)
4	Annunciation Interval	Unsigned16	68	2	the interval of the annunciation message sent out by the device
5	Duplicate Tag Detected	Boolean	70	1	this property describes if the device's PD_Tag collides with the other devices' PD_Tag (i.e. duplicated PD_Tag). TRUE= PD_Tags Collide
6	Device Redundancy Number	Unsigned8	71	1	the redundancy number of the local device, if its active device, then this value is 0, and does not have the following parameters
7	LAN Redundancy Port	Unsigned16	72	2	the LAN redundancy message processing port of the device which requests the service
8	Device Redundancy State	Unsigned8	74	1	the redundancy status of the local device: 0: active state 1: backup state If the redundancy number is 0, then the response primitive does not contain this parameter
9	Max Redundancy Number	Unsigned8	75	1	the maximum redundancy number of the device
10	Active IP Address	Unsigned32	76	4	IP address of the active device (if no redundancy, then its local IP address); if the redundancy number is 0, then the response primitive does not contain this parameter

5.3.5.2 Positive response primitive

The EM_ConfiguringDevice positive response parameters are coded as shown in Table 47.

Table 47 – Encoding of EM_ConfiguringDevice positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Destination IP Address	Unsigned32	0	4	Destination IP Address
2	Max Redundancy Number	Unsigned8	4	1	the maximum redundancy number of the device, if the redundancy number is 0, the response primitive does not contain this parameter

5.3.5.3 Negative response primitive

The EM_ConfiguringDevice negative response parameters are coded as shown in Table 48.

Table 48 – Encoding of EM_ConfiguringDevice negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Destination IP Addres	Unsigned32	0	4	IP address of the target device
2	Error Type	ErrorType	4	N	see Error Type

5.3.6 EM_SetDefaultValue service

5.3.6.1 Request primitive

The EM_SetDefaultValue request parameters are coded as shown in Table 49.

Table 49 – Encoding of EM_SetDefaultValue request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationIPAddress	Unsigned32	0	4	Destination IP address
2	Device ID	VisibleString	4	32	ID of the local device
3	PD_Tag	VisibleString	36	32	the local physical device tag, its length is 32 octets, and the unused part is padded with BLANK (0x20)

5.3.6.2 Positive response primitive

The EM_SetDefaultValue positive response parameters are coded as shown in Table 50.

Table 50 – Encoding of EM_SetDefaultValue positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Destination IP Address	Unsigned32	0	4	Destination IP address

5.3.6.3 Negative response primitive

The EM_SetDefaultValue negative response parameters are coded as shown in Table 51.

Table 51 – Encoding of clear device attribute service refuse packet

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	Destination IP Address	Unsigned32	0	4	Destination IP address
2	Error Type	ErrorType	4	N	see Error Type

5.4 Encoding of AAE Services

5.4.1 DomainDownload Service

5.4.1.1 Request primitive

The DomainDownload request parameters are coded as shown in Table 52.

Table 52 – Encoding of DomainDownload request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of the source device
2	DestinationAppID	Unsigned16	2	2	application ID of destination device
3	DestinationObjectID	Unsigned16	4	2	domain object ID of the destination device
4	DataNumber	Unsigned16	6	2	download times
5	MoreFollows	Boolean	8	1	flag used to signify if there are more downloads following
6	Reserved	OctetSring	9	1	reserved
7	DataLength	Unsigned16	10	2	data length, The range is from 0 to 512
8	Load Data	OctetString	12	N	domain download data

5.4.1.2 Positive response primitive

The DomainDownload positive response parameters are coded as shown in Table 53.

Table 53 – Encoding of domain download service response packet

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	destination IP address

5.4.1.3 Negative response primitive

DomainDownload negative response parameters are coded as shown in Table 54.

Table 54 – Encoding of DomainDownload negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	destination IP address
2	Reserved	Octetstring	2	2	reserved
3	ErrorType	ErrorType	4	N	see Error Type

5.4.2 Domain Upload Service

5.4.2.1 Request primitive

The DomainUpload request parameters are coded as shown in Table 55.

Table 55 – Encoding of DomainUpload request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of the source device
2	DestinationAppID	Unsigned16	2	2	application ID of destination device
3	DestinationObjectID	Unsigned16	4	2	domain object ID of the destination device
4	DataNumber	Unsigned16	6	2	download times

5.4.2.2 Positive response primitive

The DomainUpload positive response parameters are coded as shown in Table 56.

Table 56 – Encoding of DomainUpload positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	DataLength	Unsigned16	2	2	data length, the range is from 0 to 512
3	MoreFollows	Boolean	4	1	flag used to indicate whether there are more downloads data
4	Reserved	Octetstring	5	3	reserved
5	Load Data	Octetstring	8	N	domain upload data

5.4.2.3 Negative response primitive

The DomainUpload negative response parameters are coded as shown in Table 57.

Table 57 – Encoding of DomainUpload negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.3 EventRoport Service

The EventRoport request parameters are coded as shown in Table 58.

Table 58 – Encoding of EventRoport request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	SourceAppID	Unsigned16	2	2	application ID of source device
3	SourceObjectID	Unsigned16	4	2	domain object ID of source device
4	EventNumber	Unsigned16	6	2	number of the event
5	EventData	OctetString	8	N	specific event data

5.4.4 EventRoportAcknowledge Service

5.4.4.1 Request primitive

The EventRoportAcknowledge request parameters are coded as shown in Table 59.

Table 59 – Encoding of EventRoportAcknowledge request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	DestinationObjectID	Unsigned16	2	2	object ID of destination device
3	EventNumber	Unsigned16	4	2	event number

5.4.4.2 Positive response primitive

The EventRoportAcknowledge positive response parameters are coded as shown in Table 60.

Table 60 – Encoding of EventRoportAcknowledge positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device

5.4.4.3 Negative response primitive

The EventRoportAcknowledge negative response parameters are coded as shown in Table 61.

Table 61 – Encoding of EventRoportAcknowledge negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.5 ReportConditionChanging service

5.4.5.1 Request primitive

The ReportConditionChanging request parameters are coded as shown in Table 62.

Table 62 – Encoding of ReportConditionChanging request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	DestinationObjectID	Unsigned16	2	2	object ID of destination device
3	Enabled	Boolean	4	1	enable flag
4	Reserved	Octetstring	5	3	reserved

5.4.5.2 Positive response primitive

The ReportConditionChanging positive response parameters are coded as shown in Table 63.

Table 63 – Encoding of ReportConditionChanging positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device

5.4.5.3 Negative response primitive

The ReportConditionChanging negative response parameters are coded as shown in Table 64.

Table 64 – Encoding of ReportConditionChanging negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.6 Read service

5.4.6.1 Request primitive

The Read request parameters are coded as shown in Table 65.

Table 65 – Encoding of Read request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	DestinationObjectID	Unsigned16	2	2	object ID of destination device
3	SubIndex	Unsigned16	4	2	SubIndex of the accessed object

5.4.6.2 Positive response primitive

The Read positive response parameters are coded as shown in Table 66.

Table 66 – Encoding of Read positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Data	Octetstring	4	N	returned data

5.4.6.3 Negative response primitive

The Read negative response parameters are coded as shown in Table 67.

Table 67 – Encoding of Read negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.7 Write Service

5.4.7.1 Request primitive

The Write request parameters are coded as shown in Table 68.

Table 68 – Encoding of Write request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	DestinationObjectID	Unsigned16	2	2	object ID of destination device
3	SubIndex	Unsigned16	4	2	SubIndex of the written object
4	Reserved	Octetstring	6	2	reserved
5	Data	Octetstring	8	N	data written

5.4.7.2 Positive response primitive

The Write positive response parameters are coded as shown in Table 69.

Table 69 – Encoding of Write positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device

5.4.7.3 Negative response primitive

The Write negative response parameters are coded as shown in Table 70.

Table 70 – Encoding of Write negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.8 VariableDistribute Service

The VariableDistribute request parameters are coded as shown in Table 71.

Table 71 – Encoding of VariableDistribute request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of the source device
2	SourceObjectID	Unsigned16	2	2	object ID of the source device
3	Data	Octetstring	4	N	VariableDistributed data

5.4.9 FRTRead service

5.4.9.1 Request primitive

The FRTRead request parameters are coded as shown in Table 72.

Table 72 – Encoding of FRTRead request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationObjectID	Unsigned16	0	2	object ID of destination device
2	SubIndex	Unsigned16	2	2	SubIndex of the accessed object

5.4.9.2 Positive response primitive

The FRTRead positive response parameters are coded as shown in Table 73.

Table 73 – Encoding of FRTRead positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationObjectID	Unsigned16	0	2	object ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Data	Octetstring	4	N	returned data

5.4.9.3 Negative response primitive

The FRTRead negative response parameters are coded as shown in Table 74.

Table 74 – Encoding of FRTRead negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationObjectID	Unsigned16	0	2	object ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.10 FRTWrite Service

5.4.10.1 Request primitive

The FRTWrite request parameters are coded as shown in Table 75.

Table 75 – Encoding of FRTWrite request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationObjectID	Unsigned16	0	2	object ID of destination device
2	SubIndex	Unsigned16	2	2	SubIndex of the written object
3	Reserved	Octetstring	4	2	reserved
4	Data	Octetstring	6	N	data written

5.4.10.2 Positive response primitive

The FRTWrite positive response parameters are coded as shown in Table 76.

Table 76 – Encoding of FRTWrite positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationOjbectID	Unsigned16	0	2	object ID of destination device

5.4.10.3 Negative response primitive

The FRTWrite negative response parameters are coded as shown in Table 77.

Table 77 – Encoding of FRTWrite negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationOjbectID	Unsigned16	0	2	object ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.11 FRTVariableDistribute Service

The FRTVariableDistribute request parameters are coded as shown in Table 78.

Table 78 – Encoding of FRTVariableDistribute request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceObjectID	Unsigned16	0	2	object ID of the source device
2	Data	Octetstring	2	N	VariableDistributed data

5.4.12 BlockTransmissionOpen service

5.4.12.1 Request primitive

The BlockTransmissionOpen request parameters are coded as shown in Table 79.

Table 79 – Encoding of BlockTransmissionOpen request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of source device
2	DestinationAppID	Unsigned16	2	2	application ID of destination device
3	DestinationObjectID	Unsigned16	4	2	object ID of destination device

4	BlockType	Unsigned16	6	2	the type of block data
					0: video;
					1: audio;
					Other: Reserved

5	BlockConfigInfo	OctetString	8	N	the configuration information of the block transmission

5.4.12.2 Positive response primitive

The BlockTransmissionOpen positive response parameters are coded as shown in Table 80.

Table 80 – Encoding of BlockTransmissionOpen positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device

5.4.12.3 Negative response primitive

The BlockTransmissionOpen negative response parameters are coded as shown in Table 81.

Table 81 – Encoding of BlockTransmissionOpen negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.13 BlockTransmissionClose service

5.4.13.1 Request primitive

The BlockTransmissionClose request parameters are coded as shown in Table 82.

Table 82 – Encoding of BlockTransmissionClose request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of source device
2	DestinationAppID	Unsigned16	2	2	application ID of destination device
3	DestinationObjectID	Unsigned16	4	2	object ID of destination device

4	BlockType	Unsigned16	6	2	the type of block data
					0: video;
					1: audio;
					Other: Reserved

5.4.13.2 Positive response primitive

The BlockTransmissionClose positive response parameters are coded as shown in Table 83.

Table 83 – Encoding of BlockTransmissionClose positive response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device

5.4.13.3 Negative response primitive

The BlockTransmissionClose negative response parameters are coded as shown in Table 84.

Table 84 – Encoding of BlockTransmissionClose negative response parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	DestinationAppID	Unsigned16	0	2	application ID of destination device
2	Reserved	Octetstring	2	2	reserved
3	Error Type	ErrorType	4	N	see Error Type

5.4.14 BlockTransmit Service

The BlockTransmit request parameters are coded as shown in Table 85.

Table 85 – Encoding of BlockTransmit request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of source device
2	DestinationAppID	Unsigned16	2	2	application ID of destination device
3	DestinationObjectID	Unsigned16	4	2	object ID of destination

No.	Parameter name	Data type	Octet offset	Octet length	Description
					device
4	DataLength	Unsigned16	6	2	Data length
5	BlockType	Unsigned16	8	2	The type of block data
6	SequenceNumber	Unsigned16	10	2	Sequence number
7	TimeStamp	PrecisionTimeDifference	12	8	Time stamp
8	SendCount	Unsigned16	20	2	Total number of sent packets
9	BlockData	Octetstring	22	N	The block data

5.4.15 BlockTransmissionHeartbeat Service

The BlockTransmissionHeartbeat request parameters are coded as shown in Table 86.

Table 86 – Encoding of BlockTransmissionHeartbeat request parameters

No.	Parameter name	Data type	Octet offset	Octet length	Description
1	SourceAppID	Unsigned16	0	2	application ID of source device
2	DestinationAppID	Unsigned16	2	2	application ID of destination device
3	DestinationObjectID	Unsigned16	4	2	object ID of destination device
4	ReceptionCount	Unsigned16	6	2	the total number of the received packets
5	CumulativeLost	Unsigned16	8	2	the total number of block data packets that have been lost
6	Jitter	PrecisionTimeDifference	10	8	the transmission jitter calculated from the time stamp

6 Structure of FAL protocol state machines

Interface to FAL services and protocol machines are specified in Clause 6. Conventions used for the descriptions are given in the generic part of this standard.

Figure 2 illustrates the relationships of primitives. The primitives exchanged between each other will be described as follows.

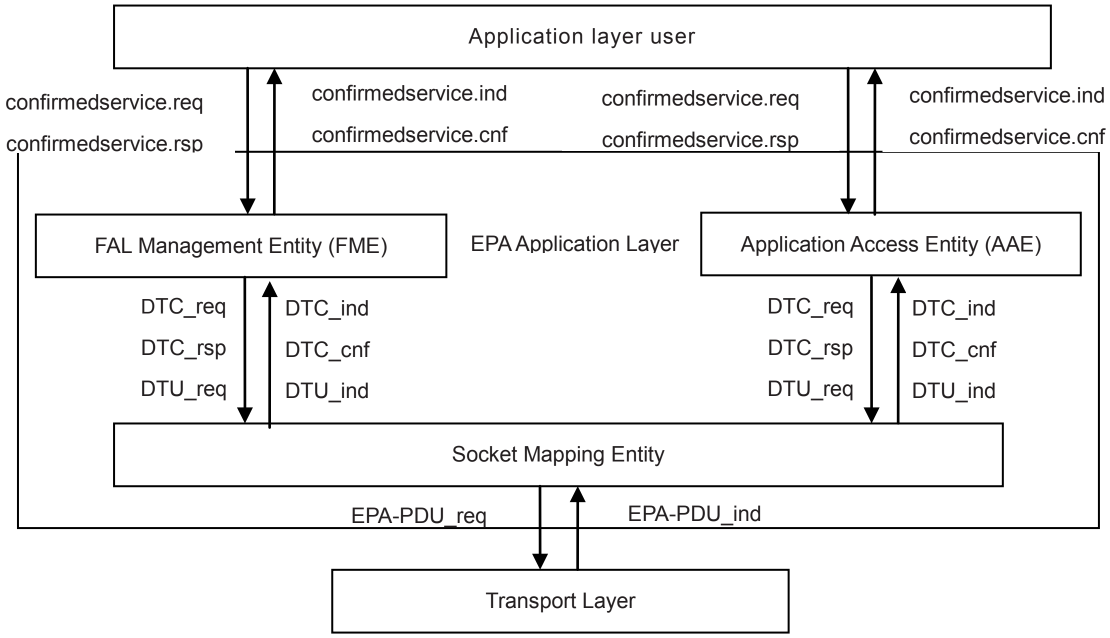

Figure 2 – Exchanged primitives of protocol state machine

7 AP-Context state machine

7.1 Primitives exchanged between ALU and ALE

Table 87 and Table 88 show the primitives exchanged between the Application Layer User (ALU) and Application Layer Entity (ALE).

Table 87 – Primitives delivered by ALU to ALE

Primitive name	Source	Associated parameters	Functions
ConfirmedService.req	ALU	Data, Remote_IP_Address	This primitive is used to send a confirmed service request primitive to ALE by the user of application layer
ConfirmedService.rsp	ALU	Data, Remote_IP_Address	This primitive is used to send a confirmed service response primitive to ALE by the user of application layer
UnconfirmedService.req	ALU	Data, Remote_IP_Address	This primitive is used to send an unconfirmed service request primitive to ALE by the user of application layer

Table 88 – Primitives delivered by ALE to ALU

Primitive name	Source	Associated parameters	Functions
ConfirmedService.ind	ALE	Data, Remote_IP_Address	This primitive is used to send a confirmed service indication primitive to the user of application layer by ALE
ConfirmedService.cnf	ALE	Data, Remote_IP_Address	This primitive is used to send a confirmed service confirmation primitive to the user of application layer by ALE
UnconfirmedService.ind	ALE	Data, Remote_IP_Address	This primitive is used to send an unconfirmed service indication primitive to the user of application layer by ALE

7.2 Protocol state machine descriptions

The AP Context Entity of Type 14 is always active. Its state is described in Table 89.

Table 89 – ACE state descriptions

States	Descriptions
ACTIVE	The ACEs in ACTIVE state prepare to transfer service primitives to ALU and FME(or AAE), or to receive primitives from ALU and FME(or AAE)

7.3 State transitions

The state transitions of ACE are defined in Figure 3, Table 90 and Table 91.

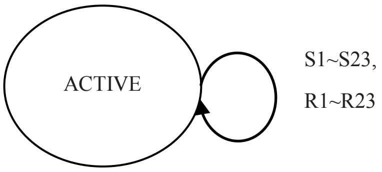

Figure 3 – ACE protocol state machine

Table 90 – ACE state transitions (sender)

#	Current state	Event or condition => action	Next state

S1	ACTIVE	DomainDownload.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S2	ACTIVE	DomainUpload.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S3	ACTIVE	AcknowlodgeEventNotification.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S4	ACTIVE	AlterEventConditionMonitor.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S5	ACTIVE	Read.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S6	ACTIVE	Write.req => confirmedservice.req { user_data := Data, Destination_ip : = remote_ip_address,	ACTIVE

#	Current state	Event or condition => action	Next state
		}	

S7	ACTIVE	EM_GetDeviceAttribute.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S8	ACTIVE	EM_SetDeviceAttribute.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S9	ACTIVE	EM_ClearDeviceAttribute.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S10	ACTIVE	DomainDownload.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S11	ACTIVE	DomainUpload.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S12	ACTIVE	AcknowlodgeEventNotification.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S13	ACTIVE	AlterEventConditionMonitor.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S14	ACTIVE	Read.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S15	ACTIVE	Write.rsp => confirmedservice.rsp { user_data := Data, Destination_ip : = remote_ip_address,	ACTIVE

#	Current state	Event or condition => action	Next state
		}	

S16	ACTIVE	EM_GetDeviceAttribute.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S17	ACTIVE	EM_SetDeviceAttribute.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S18	ACTIVE	EM_ClearDeviceAttribute.rsp
		=>
		confirmedservice.rsp {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S19	ACTIVE	EventNotification.req
		=>
		unconfirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S20	ACTIVE	Distribute.req
		=>
		unconfirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S21	ACTIVE	EM_FindTagQuery.req
		=>
		unconfirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S22	ACTIVE	EM_FindTagReply.req
		=>
		unconfirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S23	ACTIVE	EM_DeviceAnnunciation.req
		=>
		unconfirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

Table 91 – ACE state transitions (receiver)

#	Current state	Event or condition => action	Next state

R1	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “Domain Download”
		=>
		DomainDownload.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R2	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “Domain Upload”
		=>
		DomainUpload.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R3	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “Acknowlodge Event Notification”
		=>
		AcknowlodgeEventNotification.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R4	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “Alter Event Condition Monitor”
		=>
		AlterEventConditionMonitor.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R5	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “Read”
		=>
		Read.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R6	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “Write”
		=>
		Write.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R7	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “EM_GetDeviceAttribute”
		=>
		EM_GetDeviceAttribute.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R8	ACTIVE	Confirmedservice.ind APServiceType(data) = “EM_SetDeviceAttribute” =>	ACTIVE

#	Current state	Event or condition => action	Next state

		EM_SetDeviceAttribute.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}	

R9	ACTIVE	Confirmedservice.ind
		APServiceType(data) = “EM_ClearDeviceAttribute”
		=>
		EM_ClearDeviceAttribute.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R10	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “Domain Download”
		=>
		DomainDownload.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R11	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “Domain Upload”
		=>
		DomainUpload.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R12	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “Acknowlodge Event Notification”
		=>
		AcknowlodgeEventNotification.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R13	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “Alter Event Condition Monitor”
		=>
		AlterEventConditionMonitor.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R14	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “Read”
		=>
		Read.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R15	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “Write”
		=>
		Write.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

#	Current state	Event or condition => action	Next state

R16	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “EM_GetDeviceAttribute”
		=>
		EM_GetDeviceAttribute.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R17	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “EM_SetDeviceAttribute”
		=>
		EM_SetDeviceAttribute.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R18	ACTIVE	Confirmedservice.cnf
		APServiceType(data) = “EM_ClearDeviceAttribute”
		=>
		EM_ClearDeviceAttribute.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R19	ACTIVE	UnconfirmedService.ind
		APServiceType(data) = “EventNotification”
		=>
		EventNotification.ind {
		Data := user_data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R20	ACTIVE	UnconfirmedService.ind
		APServiceType(data) = “Distribute”
		=>
		Distribute.ind {
		Data := user_data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R21	ACTIVE	UnconfirmedService.ind
		APServiceType(data) = “EM_FindTagQuery”
		=>
		EM_FindTagQuery.ind {
		Data := user_data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R22	ACTIVE	UnconfirmedService.ind
		APServiceType(data) = “EM_FindTagReply”
		=>
		EM_FindTagReply.ind {
		Data := user_data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R23	ACTIVE	UnconfirmedService.ind APServiceType(data) = “EM_DeviceAnnunciation” => EM_DeviceAnnunciation.ind { Data := user_data,	ACTIVE

#	Current state	Event or condition => action	Next state
		Destination_ip : = remote_ip_address, }	

7.4 Function descriptions

Table 92 describes the functions used by ACE state transitions.

Table 92 – APServiceType() descriptions

Name	APServiceType	Used in	ACE
Input		Output	
Data		Receive the service type of service message	
Function			
To judge the message type by receiving service message, including domain download, domain upload, acknowledge event notification, alert event condition monitor, read, write, EM_GetDeviceAttribute, EM_SetDeviceAttribute, EM_ClearDeviceAttribute, Event Notification, Distribute, EM_FindTagQuery, EM_FindTagReply and EM_DeviceAnnunciation			

8 FAL management state machines

8.1 Primitives

8.1.1 Primitives exchanged between FME and application layer user

Table 93 and Table 94 show the primitives exchanged between FME and application layer user.

Table 93 – Primitives delivered by application layer user to FME

Primitive name	Source	Associated parameters	Functions
ConfirmedService.req	User of Application Layer	Data, Remote_IP_Address	This primitive is used to send a confirmed service request primitive to FME by the user of application layer
ConfirmedService.rsp	User of Application Layer	Data, Remote_IP_Address	This primitive is used to send a confirmed service response primitive to FME by the user of application layer
UnconfirmedService.req	User of Application Layer	Data, Remote_IP_Address	This primitive is used to send an unconfirmed service request primitive to FME by the user of application layer

Table 94 – Primitives delivered by FME to application layer user

Primitive name	Source	Associated parameters	Functions
ConfirmedService.ind	FME	Data, Remote_IP_Address	This primitive is used to send a confirmed service indication primitive to the user of application layer by FME
ConfirmedService.cnf	FME	Data, Remote_IP_Address	This primitive is used to send a confirmed service confirmation primitive to the user of application layer by FME
UnconfirmedService.ind	FME	Data, Remote_IP_Address	This primitive is used to send an unconfirmed service indication primitive to the user of application layer by FME

8.1.2 Primitive parameters of FME and user of application layer

Table 95 shows the primitives parameters of FME and the user of application layer.

Table 95 – Primitive parameters exchanged between FME and application layer user

Parameter name	Description
Remote_ip_address	This parameter transfers the IP address of remote device, namely the destination address sent by sender and the source address received by receiver
Data	This parameter transfers the data sent by sender and the data received by receiver

8.1.3 Primitives exchanged between FME and ESME

Table 96 and Table 97 show the primitives exchanged between FME and ESME.

Table 96 – Primitives delivered by FME to ESME

Primitive name	Source	Associated parameters	Functions
DTC_req	FME	remote_ip_address, Data	This primitive is used to send a confirmed service request primitive to ESME by FME
DTC_rsp	FME	remote_ip_address, Data	This primitive is used to send a confirmed service response primitive to ESME by FME
DTU_req	FME	remote_ip_address, Data	This primitive is used to send a unconfirmed service request primitive to ESME by FME

Table 97 – Primitives delivered by ESME to FME

Primitive name	Source	Associated parameters	Functions
DTC_ind	Type 14 Socket Mapping Entity	remote_ip_address, Data	This primitive is used to send a confirmed service indication primitive to FME by ESME
DTC_cnf	Type 14 Socket Mapping Entity	remote_ip_address, Data	This primitive is used to send a confirmed service acknowledge primitive to FME by ESME
DTU_ind	Type 14 Socket Mapping Entity	remote_ip_address, Data	This primitive is used to send a unconfirmed service indication primitive to FME by ESME

8.1.4 Primitive parameters exchanged between FME and ESME

Table 98 shows the primitives parameters exchanged between FME and ESME.

Table 98 – Primitives parameters exchanged between FME and ESME

Parameter name	Description
Remote_ip_address	This parameter transfers the IP address of remote device, namely the destination address sent by sender and the source address received by receiver
Data	This parameter transfers the data sent by sender and the data received by receiver

8.2 Protocol state machine descriptions

The states of Type 14 devices can be one of *No address*, *Unconfigured* or *Configured*. The protocol state machine is shown in Figure 4.

8.2.1 No address

In this state, the Type 14 device waits for IP address assignment. The IP address can be appointed statically by user or assigned by DHCP server. After the device gets its IP address through DHCP, it will change its next state into *Unconfigured* or *Configured* depending on its state when it was powered off before. That is, if the state when the device powered off before is *Configured*, then the next state is *Configured*, vice versa.

8.2.2 Unconfigured

In this state, the FAL Management Entity (FME) uses the specific multicast destination IP address to send EM_ActiveNotification request message to inform other devices of its presence. When this request message is received, user configuration application can query, clear or set the attributes of this device using EM_DetectingDevice, EM_ConfiguringDevice and EM_SetDefaultValue services. After configuration, FME will change its state into *Configured* and start normal operation.

8.2.3 Configured

In this state, the device can participate in normal operations. Users can configure its application information using the services provided by application layer to implement specific predefined control function.

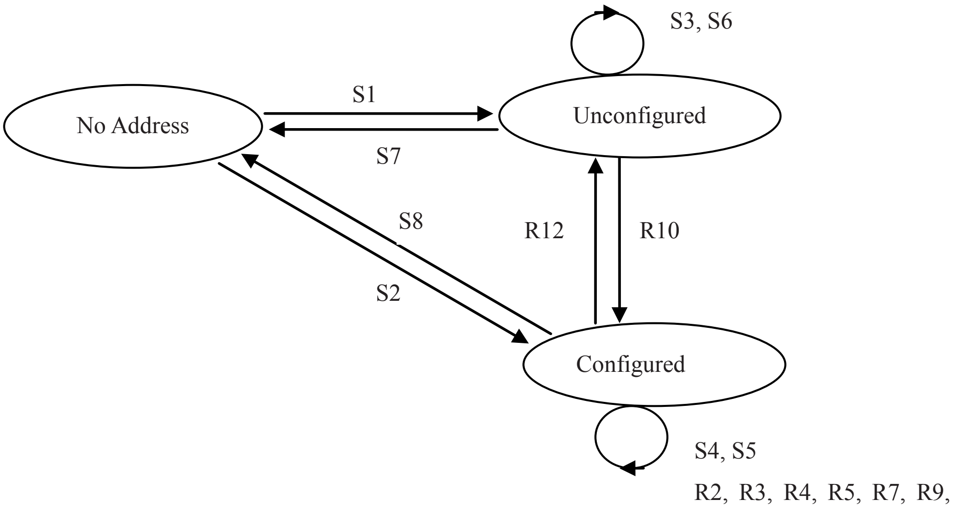

Figure 4 – FME protocol state machine

8.3 State transitions

The state transitions of FME are defined in Figure 4 and Table 99.

Table 99 – State transitions of Type 14 FME

#	Current state	Event or condition => action	Next state
S1	No Address	RcvNewIpAddress (address) = TRUE && Attribute_Set ()= FALSE => RestoreDefaults () NewAddress(address) EM_ActiveNotification.req {} Restart_Type 14RepeatTimer()	Unconfigured
S2	No Address	RcvNewIpAddress (address) = TRUE && Attribute_Set()=TRUE => Clear_DuplicatePdTagFlag() NewAddress (address) EM_ActiveNotification.req {} Restart_Type 14RepeatTimer () EM_DetectingDevice.req {}	Configured
S3	Unconfigured	RepeatTimerExpires () =>	Unconfigured

#	Current state	Event or condition => action	Next state
		EM_ActiveNotification.req {} Restart_Type 14RepeatTimer ()	

S4	Configured	EM_UnconfirmedService.req {}
		|| EM_ConfirmedService.req {}
		|| EM_ConfirmedService.rsp {}
		=>
		Send_EM_ReqRspMessage (em_svc)
			Configured

S5	Configured	EM_ConfirmedService.err {} => Send_EM_CommonErrorRsp ()	Configured
S6	Unconfigured	SntpSyncLost () => EM_ActiveNotification.req {} Restart_Type 14RepeatTimer ()	Unconfigured
S7	Unconfigured	IPAddressCollision () = TRUE = > (no actions taken)	No Address
S8	Configured	IPAddressCollision () = TRUE = > (no actions taken)	No Address
R1	Unconfigured	RecvMsg () =”Any Confirmed Type 14_FME Rsp Message” || RecvMsg ()=”Any Confirmed Type 14_FME Error Message” => EM_ConfirmedService.cnf{}	Unconfigured
R2	Configured	RecvMsg()=” EM_DetectingDevice” && QueryMatch (em_svc) = TRUE => EM_OnlineReply.req {}	Configured
R3	Configured	RecvMsg()=” EM_OnlineReply” && _MessageIdMatch(em_svc) = TRUE && DeviceId_Match (em_svc) = FALSE => Type 14Set_DuplicatePdTagFlag () EM_ActiveNotification.req {} Restart_Type 14RepeatTimer ()	Configured
R4	Configured	RecvMsg()=” EM_OnlineReply” && _MessageIDMatch(em_svc) = TRUE && DeviceId_Match (em_svc) = TRUE => //Do nothing-the response is from this device	Configured
R5	Configured	RecvMsg() = ” EM_OnlineReply” => EM_OnlineReply.ind {}	Configured
R6	Unconfigured	RecvMsg() = ” EM_GetDeviceAttributeReq” => EM_ConfrimedService.err {}	Unconfigured
R7	Configured	RecvMsg()=” EM_GetDeviceAttributeReq” => EM_GetDeviceAttribute.rsp {}	Configured

#	Current state	Event or condition => action	Next state
R8	Unconfigured	RecvMsg()=” EM_ConfiguringDeviceReq” || RecvMsg()=” EM_SetDefaultValueReq” && DeviceId_Match(em_svc)= FALSE => EM_ConfirmedService.err { }	Unconfigured
R9	Configured	RecvMsg()=”EM_ConfiguringDeviceReq” || RecvMsg()=” EM_SetDefaultValueReq” && PdTag_Match(em_svc)= TRUE => EM_ConfirmedService.rsp {}	Configured

R10	Unconfigured	RecvMsg()=” EM_ConfiguringDeviceReq”
		=>
		Set_Attribute_Data(em_svc)
		Clear_DuplicatePdTagFlag ()
		EM_ConfiguringDevice.rsp {}
		EM_ActiveNotification.req {}
		Restart_Type 14RepeatTimer ()
		EM_DetectingDevice.req {}
			Configured

R11	Configured	RecvMsg=” EM_GetDeviceAttributeReq”
		=>
		EM_GetDeviceAttribute.rsp {}
			Configured

R12	Configured	RecvMsg()=”EM_SetDefaultValueReq” => ResoreDefaults () EM_SetDefaultValue.rsp {} EM_ActiveNotification.req {} Restart_Type 14RepeatTimer ()	Unconfigured

8.4 Function descriptions

The functions used in FME state transitions are listed in Table 100 through Table 117.

NOTE The em_svc represents the message that comes from user configuration programs.

8.4.1 RcvNewIpAddress()

The RcvNewIpAddress() is illustrated in Table 100.

Table 100 – RcvNewIpAddress() descriptions

Name	RcvNewIpAddress	Using	FME
Input		Output	
Address		TRUE or FALSE	
Function			
This function is invoked when an IP address is received. The input parameters identify the interface of the device and IP address received. If this function is invoked correctly, then it returns TRUE, otherwise it returns FALSE			

8.4.2 Attribute_Set()

The Attribute_Set() is illustrated in Table 101.

Table 101 – Attribute_Set() descriptions

Name	Attribute_Set	Using	FME
Input		Output	
None		TRUE or FALSE	
Function			
Returns TRUE if the attribute of the Type 14 device has been set by the EM_ConfiguringDevice service and is currently still valid. Otherwise, returns FALSE			

8.4.3 RestoreDefaults()

The RestoreDefaults() is illustrated in Table 102.

Table 102 – RestoreDefaults() descriptions

Name	RestoreDefaults	Using	FME
Input		Output	
None		None	
Function			
When this function is invoked, all links are disconnected, and the attributes of EME are set as default value			

8.4.4 NewAddress()

The NewAddress() is illustrated in Table 103.

Table 103 – NewAddress() descriptions

Name	NewAddress	Using	FME
Input		Output	
Address		TRUE or FALSE	
Function			
Returns TRUE if the IP address is updated correctly when Type 14 device received a new IP address. Otherwise, returns FALSE			

8.4.5 Restart_Type 14RepeatTimer()

Restart_Type 14RepeatTimer() is illustrated in Table 104.

Table 104 – Restart_Type 14RepeatTimer() descriptions

Name	Restart_Type 14RepeatTimer	Using	FME
Input		Output	
None		None	
Function			
This function is invoked to restore and restart Type 14 RepeatTimer			

8.4.6 Clear_DuplicatePdTagFlag()

The Clear_DuplicatePdTagFlag() is illustrated in Table 105.

Table 105 – Clear_DuplicatePdTagFlag() descriptions

Name	Clear_DuplicatePdTagFlag	Using	FME
Input		Output	
None		None	
Function			
This function is invoked to clear duplicate detected state.			

8.4.7 Type 14RepeatTimerExpire()

The Type 14RepeatTimerExpire() is illustrated in Table 106.

Table 106 – Type 14RepeatTimerExpire() descriptions

Name	Type 14RepeatTimerExpire	Using	FME
Input		Output	
None		None	
Function			
This function is invoked when the Type 14 Repeat Timer expires			

8.4.8 Send_EM_ReqRspMessage()

The Send_EM_ReqRspMessage() is illustrated in Table 107.

Table 107 – Send_EM_ReqRspMessage() descriptions

Name	Send_EM_ReqRspMessage	Using	FME
Input		Output	
Em_svc		None	
Function			
Constructs and sends request or response messages			

8.4.9 Send_EM_CommonErrorRsp()

The Send_EM_CommonErrorRsp() is illustrated in Table 108.

Table 108 – Send_EM_CommonErrorRsp() descriptions

Name	Send_EM_CommonErrorRsp	Using	FME
Input		Output	
service_type, Spec_params		None	
Function			
Constructs and sends error response information			

8.4.10 SntpSyncLost()

The SntpSyncLost() is illustrated in Table 109.

Table 109 – SntpSyncLost() descriptions

Name	SntpSyncLost	Using	FME
Input		Output	
None		None	
Function			
This function is invoked when the state of synchronization of time between the device and the selected remote time server changes from synchronization to no–synchronization			

8.4.11 IPAddressCollision()

The IPAddressCollision() is illustrated in Table 110.

Table 110 – IPAddressCollision() descriptions

Name	IPAddressCollision	Using	FME
Input		Output	
None		TRUE or FALSE	
Function			
If the IP address is conflict with others, returns TURE. Otherwise, returns FALSE			

8.4.12 RecvMsg()

The RecvMsg() is illustrated in Table 111.

Table 111 – RecvMsg() descriptions

Name	RecvMsg	Using	FME
Input		Output	
Em_svc		Em_svc message type	
Function			
This function is invoked when a message is received. It decodes the em_svc and returns the type of the Type 14 management service			

8.4.13 QueryMatch()

The QueryMatch() is illustrated in Table 112.

Table 112 – QueryMatch() descriptions

Name	QueryMatch	Using	FME
Input		Output	
Em_svc		TRUE or FALSE	
Function			
Returns TRUE if the queried object is contained in the device			

8.4.14 MessageIDMatch()

The MessageIDMatch() is illustrated in Table 113.

Table 113 – MessageIDMatch() descriptions

Name	MessageIDMatch	Using	FME
Input		Output	
Em_svc		TRUE or FALSE	
Function			
Returns TRUE if the MessageID contained in the messages matches that contained in the EM_DetectingDevice service			

8.4.15 DeviceId_Match()

The DeviceId_Match() is illustrated in Table 114.

Table 114 – DevId_Match() descriptions

Name	DeviceId_Match	Using	FME
Input		Output	
em_svc		TRUE or FALSE	
Function			
Returns TRUE if the Device ID in the message exactly matches the Device ID of this device			

8.4.16 PdTag_Match()

The PdTag_Match() is illustrated in Table 115.

Table 115 – PdTag_Match() descriptions

Name	PdTag_Match	Using	FME
Input		Output	
em_svc		TRUE or FALSE	
Function			
Returns TRUE if the PD_Tag contained in the message matches the PD_Tag of the device			

8.4.17 Set_Attribute_Data()

The Set_Attribute_Data() is illustrated in Table 116.

Table 116 – Set_Attribute_Data() descriptions

Name	Set_Attribute_Data	Using	FME
Input		Output	
em_svc		None	
Function			
Updates the attributes of the EME in the Type 14 device			

8.4.18 Set_DuplicatePdTagFlag()

The Set_DuplicatePdTagFlag() is illustrated in Table 117.

Table 117 – Set_DuplicatePdTagFlag() descriptions

Name	Set_DuplicatePdTagFlag	Using	FME
Input		Output	
None		None	
Function			
The value of DuplicateDetectedState is set if the device has detected that another device has the same PD_Tag			

9 Application access entity protocol machine

9.1 Primitives

9.1.1 Primitives exchanged between AAE and application layer user

The primitives exchanged between AAE and Application Layer User (ALU) are shown in Table 118 and Table 119.

Table 118 – Primitives issued by ALU to AAE

Primitive name	Source	Associated parameters	Functions
ConfirmedService.req	ALU	remote_ip_address, data	This is a primitive used to convey a ConfirmedService request primitive from the ALU to the AAE
ConfirmedService.rsp	ALU	remote_ip_address, data	This is a primitive used to convey a ConfirmedService response primitive from the ALU to the AAE
UnconfirmedService.req	ALU	remote_ip_address, data	This is a primitive used to convey a UnconfirmedService request primitive from the ALU to the AAE

Table 119 – Primitives issued by AAE to ALU

Primitive name	Source	Associated parameters	Functions
ConfirmedService.ind	AAE	remote_ip_address, data	This is a primitive used to convey a ConfirmedService indication primitive from the AAE to the ALU
ConfirmedService.cnf	AAE	remote_ip_address, data	This is a primitive used to convey a ConfirmedService confirmation primitive from the AAE to the ALU
UnconfirmedService.ind	AAE	remote_ip_address, data	This is a primitive used to convey a UnconfirmedService indication primitive from the AAE to the ALU

9.1.2 Primitive parameters exchanged between AAE and ALU

Table 120 describes the parameters used with primitives exchanged between AAE and ALU.

Table 120 – Primitives parameters exchanged between AAE and ALU

Parameter name	Description
remote_ip_address	This parameter transfers the IP address of the remote device, namely the destination address that the sender will send data to and the source address that the receiver will receive data from
Data	This parameter transfers the data that the sender will send and the receiver will receive

9.1.3 Primitives Exchanged between AAE and ESME

The primitives exchanged between AAE and ESME are shown in Table 121 and Table 122.

Table 121 – Primitives issued by AAE to ESME

Primitive name	Source	Associated parameters	Functions
DTC.req	AAE	remote_ip_address, data	This is a primitive used to convey a ConfirmedService request primitive from the AAE to the ESME
DTC.rsp	AAE	remote_ip_address, data	This is a primitive used to convey a ConfirmedService response primitive from the AAE to the ESME
DTU.req	AAE	remote_ip_address, data	This is a primitive used to convey a UnconfirmedService request primitive from the AAE to the ESME

Table 122 – Primitives issued by ESME to AAE

Primitive name	Source	Associated parameters	Functions
DTC.ind	ESME	remote_ip_address, data	This is a primitive used to convey a ConfirmedService indication primitive from the ESME to the AAE
DTC.cnf	ESME	remote_ip_address, data	This is a primitive used to convey a ConfirmedService confirmation primitive from the ESME to the AAE
DTU.ind	ESME	remote_ip_address, data	This is a primitive used to convey a UnconfirmedService indication primitive from the ESME to the AAE

9.1.4 Primitive parameters exchanged between AAE and ESME

Table 123 describes the parameters used with primitives exchanged between AAE and ESME.

Table 123 – Primitive parameters exchanged between AAE and ESME

Parameter name	Description
remote_ip_address	This parameter transfers the IP address of the remote device; namely, the destination address that the sender will send data to and the source address that the receiver will receive data from
Data	This parameter transfers the data that the sender will send and the receiver will receive

9.2 AAE state machine

9.2.1 AAE state description

Type 14 Application Access Entity is always active. Its state is described in Table 124.

Table 124 – AAE state descriptions

States	Descriptions
ACTIVE	The ACEs in ACTIVE state prepare to transfer service primitives to ALU and ESME, or to receive primitives from ALU and ESME

9.2.2 State transitions

The protocol state machine of AAE is illustrated in the Figure 5 and Table 125 through Table 126.

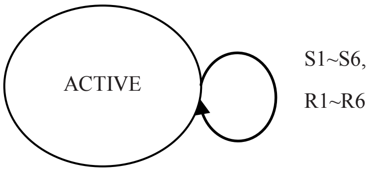

Figure 5 – AAE state transition diagrams

Table 125 – AAE state transitions (sender)

#	Current state	Event or condition => action	Next state

S1	ACTIVE	confirmedservice.req
		=>
		confirmedservice.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S2	ACTIVE	confirmedservice.rsp
		=>
		confirmedservice.rsp {
		user_data := Data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S3	ACTIVE	unconfirmedservice.req
		=>
		unconfirmedservice .req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S4	ACTIVE	ConfirmedService.req => DTC.req {	ACTIVE

#	Current state	Event or condition => action	Next state

		user_data := Data,
		Destination_ip : = remote_ip_address,
		}	

S5	ACTIVE	ConfirmedService.rsq
		=>
		DTC.rsq {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

S6	ACTIVE	UnconfirmedService.req
		=>
		DTU.req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

Table 126 – AAE state transitions (receiver)

#	Current state	Event or condition => action	Next state

R1	ACTIVE	Confirmedservice.ind
		=>
		ConfirmedService.ind {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R2	ACTIVE	Confirmedservice.cnf
		=>
		ConfirmedService.cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R3	ACTIVE	UnconfirmedService.ind
		=>
		UnconfirmedService.ind {
		Data := user_data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R4	ACTIVE	DTC.ind
		&& ServiceType(data) = “Confirmed Service Indication”
		=>
		ConfirmedService.ind {
		Receivedata := data,
		Remote_ip : = remote_ip_address,
		}
			ACTIVE

R5	ACTIVE	DTC.cnf
		&& ServiceType(data) = “Confirmed Service Confirmation”
		=>
		ConfirmedService.cnf {
		Receivedata := data,
		Remote_ip : = remote_ip_address,
		}
			ACTIVE

R6	ACTIVE	DTU.ind && ServiceType(data) = “Unconfirmed Service Indication”	ACTIVE

#	Current state	Event or condition => action	Next state

		=>
		UnconfirmedService.ind {
		Receivedata := data,
		Remote_ip : = remote_ip_address,
		}	

9.2.3 Function descriptions

Table 127 describes the functions used by AAE state transitions.

Table 127 – ServiceType() descriptions

Name	ServiceType	Used in	AAE
Input		Output	
Data		Receive the primitive type of service message	
Function			
To judge the message type by receiving service message, including confirm service indication primitive, confirm service primitive and non-confirm service indication primitive			

9.3 Event ASE protocol machine

9.3.1 State description

Event ASE has two states: LOCKED and UNLOCKED described in Table 128.

Table 128 – State value of event management

States	Descriptions
LOCKED	LOCKED(Enabled=FALSE)means no event notification to transfer
UNLOCKED	UNLOCKED(Enable = TRUE)means event notification to transfer normally

9.3.2 State transitions

State transitions of Event ASE are shown in Figure 6 and Table 129.

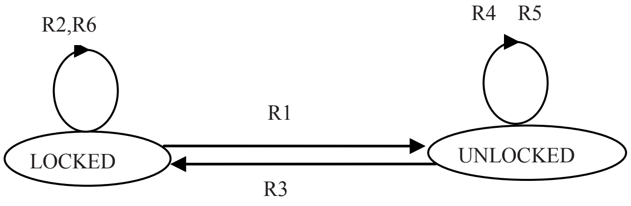

Figure 6 – Event ASE state transition diagrams

Table 129 – Event ASE state transition table

#	Current state	Event or condition => action	Next state

R1	LOCKED	ReportConditionChanging.ind
		&& Enabled = TRUE
		=>
		ReportConditionChanging.rsq(+){
		}
			UNLOCKED

R2	LOCKED	ReportConditionChanging.ind
		&&Enabled = FALSE
		=>
		ReportConditionChanging.rsq(+){
		}
			LOCKED

R3	UNLOCKED	ReportConditionChanging.ind
		&&Enabled = FALSE
		=>
		ReportConditionChanging.rsq(+){
		}
			LOCKED

R4	UNLOCKED	ReportConditionChanging.ind
		&&Enabled = TRUE
		=>
		ReportConditionChanging.rsq(+){
		}
			UNLOCKED

R5	UNLOCKED	Type 14 AL service.ind <>" ReportConditionChanging.ind" => (no actions taken)	UNLOCKED
R6	LOCKED	Type 14 AL service.ind <>" ReportConditionChanging.ind" => (no actions taken)	LOCKED

9.3.3 Function descriptions

No additional functions are used in Event ASE state transitions.

9.4 Domain ASE protocol machine

9.4.1 State descriptions

Domain ASE has five states described in Table 130.

Table 130 – Domain state value

Domain state code	value	Specification
EXISTENT	0	This domain exists but its content is not defined
DOWNLOADING	1	This domain is being downloaded
UPLOADING	2	This domain is being uploaded
READY	3	This domain can be used
IN-USE	4	This domain is in use by some program. It cannot be downloaded or uploaded in this state

9.4.2 State transitions

The state transitions of Domain ASE are illustrated in Figure 7 and Table 131.

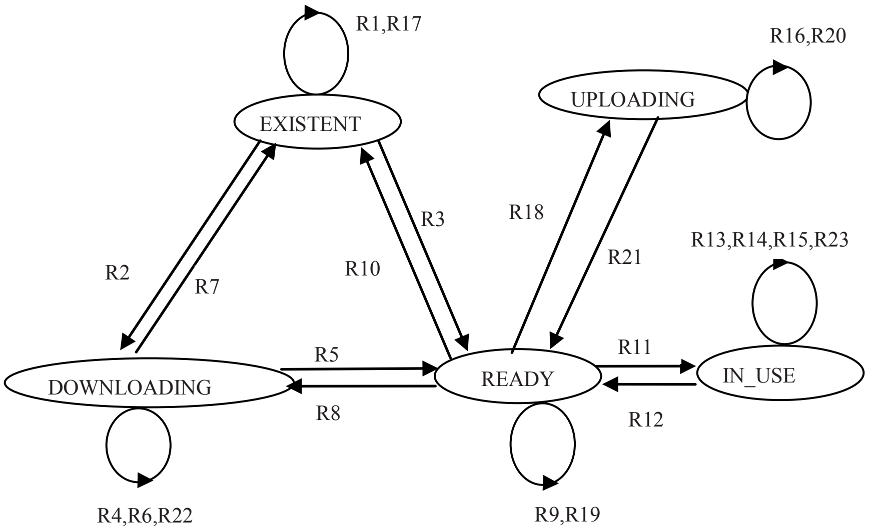

Figure 7 – Domain ASE state transition diagram

Table 131 – Domain ASE state transition table

	Current state	Event or condition => action	Next state

R1	EXISTENT	DomainDownload.ind
		&& Domain_DownloadSucceed() = FALSE
		=>
		DomainDownload.err{
		}
			EXISTENT

R2	EXISTENT	DomainDownload.ind
		&& Domain_DownloadSucceed() = TRUE
		&&MoreFollows = TRUE
		=>
		DomainDownload.rsp(+){
		}
		Domain_WriteBuffer()
			DOWNLOADING

R3	EXISTENT	DomainDownload.ind
		&& Domain_DownloadSucceed() = TRUE
		&&MoreFollow= FALSE
		=>
		DomainDownload.rsp(+){
		}
		Domain_WriteBuffer()
			READY

R4	DOWNLOADING	DomainDownload.ind && Domain_DownloadSucceed() = TRUE &&MoreFollows = TRUE => DomainDownload.rsp(+){ }. Domain_WriteBuffer()	DOWNLOADING
R5	DOWNLOADING	DomainDownload.ind && Domain_DownloadSucceed() = TRUE &&MoreFollow= FALSE => DomainDownload.rsp(+){ }. Domain_WriteBuffer()	READY
R6	DOWNLOADING	DomainDownload.ind && Domain_DownloadSucceed() = FALSE => DomainDownload.err{	DOWNLOADING

	Current state	Event or condition => action	Next state
		}	

R7	DOWNLOADING	DomainDownload.ind
		&& Domain_DownloadSucceed() = FALSE
		&& DownloadFalseCounting >3
		=>
		DomainDownload.err{
		}
			EXISTENT

R8	READY	DomainDownload.ind && Domain_DownloadSucceed() = TRUE &&MoreFollow = TRUE => DomainDownload.rsp(+){ }. Domain_WriteBuffer()	DOWNLOADING
R9	READY	DomainDownload.ind && Domain_DownloadSucceed() = TRUE &&MoreFollow = FALSE => DomainDownload.rsp(+){ }. Domain_WriteBuffer()	READY

R10	READY	DownloadSequence.ind
		&& Domain_DownloadSucceed() = FALSE
		=>
		DomainDownload.err{
		}
			EXISTENT

R11	READY	IncreamentInvokeDomainCounter()
		&& Counter=0
		=>
		Counter = 1
			IN_USE

R12	IN_USE	DecreamentInvokeDomainCounter()
		&& Counter=1
		=>
		Counter = 0
			READY

R13	IN_USE	IncreamentInvokeDomainCounter()
		=>
		Counter = Counter +1
			IN_USE

R14	IN_USE	DecreamentInvokeDomainCounter()
		&&counter>1
		=>
		Counter = Counter -1
			IN_USE

R15	IN_USE	DomainDownload.ind
		=>
		DomainDownload.err{
		}
			IN_USE

R16	UPLOADING	DomainDownload.ind
		=>
		DomainDownload.rsp(-){
		ErrorType:=Service Error
		}
			UPLOADING

R17	EXISTENT	DomainUpload.ind
		=>
		DomainDownload.rsp(-){
		ErrorType:=Service Error
		}
			EXISTENT

R18	READY	DomainUpload.ind &&MoreFollows=TRUE	UPLOADING

	Current state	Event or condition => action	Next state

		=>
		DomainUpload.rsp(+){
		MoreFollows := TRUE
		}	

R19	READY	DomainUpload.ind
		&&MoreFollows=FALSE
		=>
		DomainUpload.rsp(+){
		MoreFollows: = FALSE
		}
			READY

R20	UPLOADING	DomainUpload.ind
		&&MoreFollows=TRUE
		=>
		DomainUpload.rsp(+){
		MoreFollows = TRUE
		}
			UPLOADING

R21	UPLOADING	DomainUpload.ind
		&&MoreFollows=FALSE
		=>
		DomainUpload.rsp(+){
		MoreFollows := FALSE
		}
			READY

R22	DOWNLOADING	DomainUpload.ind
		=>
		DomainUpload.rsp(-){
		ErrorType:=Service Error
		}
			DOWNLOADING

R23	IN_USE	DomainUpload.ind
		=>
		DomainUpload.rsp (-){
		ErrorType:=Service Error
		}
			IN_USE

9.4.3 Functions description

Table 132 through Table 135 described the functions used in the Domain ASE state transitions.

9.4.3.1 Domain_DownloadSucceed() description

Table 132 describes Domain_DownloadSucceed() function.

Table 132 – Domain_DownloadSucceed() description

Name	Domain_DownloadSucceed	Used in	AAE
Input		Output	
None		TRUE or FALSE	
Function			
Judge the download state, if failed, then returns FALSE; if succeeded, then returns TRUE			

9.4.3.2 Domain_WriteBuffer() description

Table 133 describes Domain_WriteBuffer() function.

Table 133 – Domain_WriteBuffer() description

Name	Domain_WriteBuffer	Used in	AAE
Input		Output	
None		None	
Function			
Write the received data into buffer			

9.4.3.3 IncreamentInvokeDomainCounter() description

Table 134 describes IncreamentInvokeDomainCounter() function.

Table 134 – IncreamentInvokeDomainCounter() description

Name	IncreamentInvokeDomainCounter	Used in	AAE
Input		Output	
None		None	
Function			
IncreamentInvokeDomainCounter increases by 1			

9.4.3.4 DecreamentInvokeDomainCounter() description

Table 135 describes DecreamentInvokeDomainCounter() function.

Table 135 – DecreamentInvokeDomainCounter() description

Name	DecreamentInvokeDomainCounter	Used in	AAE
Input		Output	
None		None	
Function			
DecreamentInvokeDomainCounter decreases by 1			

9.5 Block ASE protocol machine

9.5.1 State description

Block ASE has two states: IDLE and TRANSMISSION described in Table 136.

Table 136 – State value of Block transmission

States	Descriptions
IDLE	The device has no block transmitting mission currently
TRANSMITTING	The device is transmitting block data currently

9.5.2 State transitions

State transitions of Block ASE are shown in Figure 8 and Table 137.

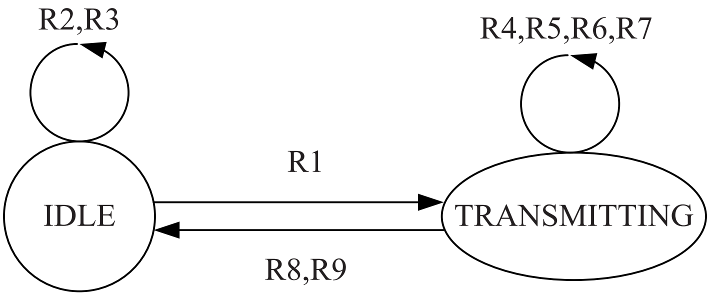

Figure 8 – Block ASE state transition diagrams

Table 137 – Block ASE state transition table

#	Current state	Event or condition => action	Next state

R1	IDLE	BlockTransmissionOpen.ind
		&& BlockTransmissionOpenSucceed() = TRUE
		&& MemberNum = 0
		=>
		BlockTransmissionOpen.rsp(+){
		}
		BlockTransmit.ind{
		}
		MemberNum = 1
			TRANSMITTING

R2	IDLE	BlockTransmissionClose.ind
		=>
		BlockTransmissionClose.rsp(-){
		ErrorType:=Service Error
		}
			IDLE

R3	IDLE	BlockTransmissionOpen.ind
		&& BlockTransmissionOpenSucceed() = FLASE
		=>
		BlockTransmissionOpen.err{
		}
			IDLE

R4	TRANSMITTING	BlockTransmissionOpen.ind
		&& BlockTransmissionOpenSucceed() = FLASE
		=>
		BlockTransmissionOpen.err{
		}
			TRANSMITTING

R5	TRANSMITTING	BlockTransmissionOpen.ind
		&& BlockTransmissionOpenSucceed() = TRUE
		&& MemberNum > 0
		=>
		BlockTransmissionOpen.rsp(+){
		}
		MemberNum = MemberNum + 1
			TRANSMITTING

R6	TRANSMITTING	BlockTransmissionClose.ind
		&& BlockTransmissionCloseSucceed() = TRUE
		&& MemberNum > 1
		=>
		BlockTransmissionClose.rsp(+){
		}
		MemberNum = MemberNum – 1
			TRANSMITTING

R7	TRANSMITTING	BlockTransmissionClose.ind && BlockTransmissionCloseSucceed() = FLASE =>	TRANSMITTING

#	Current state	Event or condition => action	Next state

		BlockTransmissionClose.err{
		}	

R8	TRANSMITTING	BlockTransmissionClose.ind
		&& BlockTransmissionCloseSucceed() = TRUE
		&& MemberNum = 1
		=>
		BlockTransmissionClose.rsp(+){
		}
		MemberNum = 0
			IDLE

R9	TRANSMITTING	ReceiveBlockTransmissionHeartbeat_timeout = TRUE
		=>
		ReceiveBlockTransmissionHeartbeat_timeout()
		MemberNum = 0
			IDLE

9.5.3 Function descriptions

Table 138 through Table 140 describes the functions used by Block ASE state transitions.

Table 138 – BlockTransmissionOpenSucceed() descriptions

Name	BlockTransmissionOpenSucceed	Used in	AAE
Input		Output	
Data		TRUE or FALSE	
Function			
Judge the BlockTransmissionOpen state, if failed, then returns FALSE; if succeeded, then returns TRUE			

Table 139 – BlockTransmissionCloseSucceed() descriptions

Name	BlockTransmissionCloseSucceed	Used in	AAE
Input		Output	
Data		TRUE or FALSE	
Function			
Judge the BlockTransmissionClose state, if failed, then returns FALSE; if succeeded, then returns TRUE			

Table 140 – ReceiveBlockTransmissionHeartbeat_timeout() description

Name	ReceiveBlockTransmissionHeartbeat_timeout	Used in	AAE
Input		Output	
None		None	
Function			
End the transmission of block data if the device hasn’t received BlockTransmissionHeartbeat in a certain time			

10 Application relationship state machine

10.1 Primitives

10.1.1 Primitives Exchanged between AREP and FME(or AAE)

The primitives exchanged between AREP and FME(or AAE) are shown in Table 141 and Table 142.

Table 141 – Primitives issued by FME(or AAE) to AREP

Primitive name	Source	Associated parameters	Functions
DTC_req	FME(or AAE)	remote_ip_address, data	This is a primitive used to convey a ConfirmedService request primitive from the FME(or AAE) to the AREP
DTC_rsp	FME(or AAE)	remote_ip_address, data	This is a primitive used to convey a ConfirmedService response primitive from the FME(or AAE) to the AREP
DTU_req	FME(or AAE)	remote_ip_address, data	This is a primitive used to convey a UnconfirmedService request primitive from FME(or AAE) to the AREP

Table 142 – Primitives issued by AREP to FME(or AAE)

Primitive name	Source	Associated parameters	Functions
DTC_ind	AREP	remote_ip_address, data	This is a primitive used to convey a ConfirmedService indication primitive from the AREP to the FME(or AAE)
DTC_cnf	AREP	remote_ip_address, data	This is a primitive used to convey a ConfirmedService confirmation primitive from the AREP to the FME(or AAE)
DTU_ind	AREP	remote_ip_address, data	This is a primitive used to convey a UnconfirmedService indication primitive from the AREP to the FME(or AAE)

10.1.2 Primitive parameters exchanged between AREP and FME(or AAE)

Table 143 describes the parameters used with primitives exchanged between AREP and FME(or AAE).

Table 143 – Primitives parameters exchanged between AREP and FME(or AAE)

Parameter name	Description
remote_ip_address	This parameter transfers the IP address of the remote device, namely the destination address that the sender will send data to and the source address that the receiver will receive data from
Data	This parameter transfers the data that the sender will send and the receiver will receive

10.1.3 Primitives Exchanged between AREP and ESME

The primitives exchanged between AREP and ESME are shown in Table 144 and Table 145.

Table 144 – Primitives issued by AREP to ESME

Primitive name	Source	Associated parameters	Functions
Type 14_PDU_req	AREP	remote_ip_address, data	This is a primitive used to convey a ConfirmedService request primitive from the AREP to the ESME

Table 145 – Primitives issued by ESME to AREP

Primitive name	Source	Associated parameters	Functions
Type 14_PDU_ind	ESME	remote_ip_address, data	This is a primitive used to convey a ConfirmedService indication primitive from the ESME to the AREP

10.1.4 Primitive parameters exchanged between AREP and ESME

Table 146 describes the parameters used with primitives exchanged between AAE and ESME.

Table 146 – Primitive parameters exchanged between AREP and ESME

Parameter name	Description
remote_ip_address	This parameter transfers the IP address of the remote device; namely, the destination address that the sender will send data to and the source address that the receiver will receive data from
Data	This parameter transfers the data that the sender will send and the receiver will receive

10.2 AREP state description

Type 14 Application Relation Endpoint is always active. Its state is described in Table 147.

Table 147 – AREP state descriptions

States	Descriptions
ACTIVE	The AREP in ACTIVE state prepares to transfer service primitives to ALU and ESME, or to receive primitives from ALU and ESME

10.3 State transitions

The protocol state machine of AREP is illustrated in the Figure 9 and Table 148.

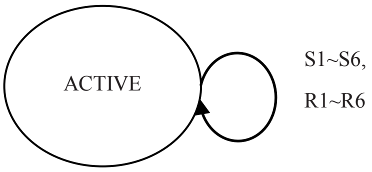

Figure 9 – AREP state transition diagrams

Table 148 – AREP state transitions

#	Current state	Event or condition => action	Next state

S1	ACTIVE	DTC_req
		|| DTC_rsp
		|| DTU_req
		=>
		Type 14_PDU_req {
		user_data := Data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R1	ACTIVE	Type 14_PDU_ind
		&& AREPType(data) = “peer”
		&& MessageType(data) = “Confirmed Service Indication”
		=>
		DTC_cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R2	ACTIVE	Type 14_PDU_ind
		&& AREPType(data) = “peer”
		&& MessageType(data) = “Confirmed Service Confirmation”
		=>
		DTC_cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

#	Current state	Event or condition => action	Next state

R3	ACTIVE	Type 14_PDU_ind
		&& AREPType(data) = “client”
		=>
		DTC_cnf {
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R4	ACTIVE	Type 14_PDU_ind
		&& AREPType(data) = “server”
		=>
		DTC_ind{
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

R5	ACTIVE	Type 14_PDU_ind && AREPType(data) = “publisher” => No action	ACTIVE

R6	ACTIVE	Type 14_PDU_ind
		&& AREPType(data) = “subscriber”
		=>
		DTU_ind{
		Data := user_data
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

10.4 Function descriptions

Table 149 describes the functions used by AREP state transitions.

Table 149 – AREPType() descriptions

Name	AREPType	Used in	AREP
Input		Output	
Data		The type of AREP	
Function			
To judge the type of AREP, including peer, client, server, publisher and subscriber			

Table 150 describes the functions used by AAE state transitions.

Table 150 – ServiceType() descriptions

Name	ServiceType	Used in	AAE
Input		Output	
Data		Receive the primitive type of service message	
Function			
To judge the message type by receiving service message, including confirm service indication primitive, confirm service primitive and non-confirm service indication primitive			

11 DLL mapping protocol machine

11.1 Concept

In Type 14 system, both UPD/IP and ISO/IEC 8802-3 protocols are applied directly for Type 14 as sublayers of DLL defined in IEC 61158-1. This clause defines the interface between

Type 14 FAL services and UDP/IP layer which is called Type 14 Socket Mapping Entity (ESME).

11.2 Primitives

11.2.1 The primitives and parameters exchanged between AAE and ESME

The primitives exchanged between AAE and ESME are described in 9.1.3.

The parameters used with the primitives between AAE and ESME are described in 9.1.4.

11.2.2 The primitives and parameters exchanged between FME and ESME

The primitives exchanged between FME and ESME are described in 8.1.3.

The parameters used with the primitives exchanged between FME and ESME are described in 8.1.4.

11.2.3 Transport Layer and ESME primitive

Table 151 describes the primitives exchanged between Transport layer (UDP) and ESME.

Table 151 – The primitives exchanged between transport layer and ESME

Primitive name	source	Reference parameters
Type 14_PDU_req	Socket mapping entity	remote_ip_address, data
Type 14_PDU_ind	Transport layer	remote_ip_address, data

11.2.4 Primitives parameters exchanged between Transport Layer and ESME

Table 152 illustrates the primitives parameters exchanged between Transport Layer and ESME.

Table 152 – Primitives parameters exchanged between Transport Layer and ESME

Parameter name	description
remote_ip_address	This parameter transfers the IP address of the remote device, namely the destination address the sending side will send to and the source address the receiving side will receive from
Data	This parameter transfers the data that the sending side will send and the receiving side will receive

11.3 State description

ESME is always active. Its state is described in Table 153.

Table 153 – ESME state description

State name	description
ACTIVE	ESME transfers primitives to AAE, FME to transport layer; or it is ready to receive primitives from AAE, FME to transport layer

11.4 State transitions

Figure 10 and Table 154 illustrates ESME state transitions.

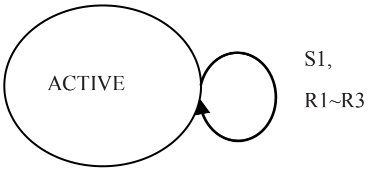

Figure 10 – ESME state transition

Table 154 – ECFME state transitions

#	Current state	Event or condition => action	Next state

	ACTIVE	DTC_req
		|| DTC_rsp
		|| DTU_req
		=>
		Type 14-PDU.req {
		Senddata := data,
		Destination_ip : = remote_ip_address,
		}
			ACTIVE

	ACTIVE	Type 14-PDU.ind
		&& ServiceType(data) = “Confirmed Service Indication”
		=>
		DTC.ind{
		Receivedata := data,
		Remote_ip : = remote_ip_address,
		}
			ACTIVE

	ACTIVE	Type 14-PDU.ind
		&& ServiceType(data) = “Confirmed Service Confirmation”
		=>
		DTC.cnf{
		Receivedata := data,
		Remote_ip : = remote_ip_address,
		}
			ACTIVE

	ACTIVE	Type 14-PDU.ind
		&& ServiceType(data) = “Unconfirmed Service Indication”
		=>
		DTU.ind{
		Receivedata := data,
		Remote_ip : = remote_ip_address,
		}
			ACTIVE

11.5 Function description

Table 155 describes the function used in ESME state transitions.

Table 155 – ServiceType()description

name	ServiceType	use	Socket mapping entity
input		output	
Data		Received primitive type of service message	
function			
Judge the message type by the received service message, including confirmed service indication primitive, confirmed service primitive and unconfirmed service indication primitive			

Bibliography

IEC 61158-1, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

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/TR 8802-1, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 1: Overview of Local Area Network Standards*

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

ISO/IEC 8859-1, *Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1*

ISO/IEC 8877, *Information technology – Telecommunications and information exchange between systems – Interface connector and contact assignments for ISDN Basic Access Interface located at reference points S and T*

ISO 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

IEEE 802.1Q, *IEEE standard for Local and metropolitan area networks – Virtual bridged local area networks*, available at <http://www.ieee.org>

____

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

All the data, software and documentation set out in all British Standards and other BSI publications are the property of and copyrighted by BSI, or some person or entity that owns copyright in the information used (such as the international standardization bodies) and has formally licensed such information to BSI for commercial publication and use. Except as permitted under the Copyright, Designs and Patents Act 1988 no extract may be reproduced, stored in a retrieval system or transmitted in any form or by any means – electronic, photocopying, recording or otherwise – without prior written permission from BSI. Details and advice can be obtained from the Copyright & Licensing Department.

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.™