IEC 61158-6-10

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications – Part 6-10: Application layer protocol specification – Type 10 elements

IEC 61158-6-10:2007(E)

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

About the IEC

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

About IEC publications

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

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.
- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.
- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.
- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

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

IEC 61158-6-10

Edition 1.0 2007-12

INTERNATIONAL STANDARD

Industrial communication networks – Fieldbus specifications –
Part 6-10: Application layer protocol specification – Type 10 elements

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE **XH**

ICS 35.100.70; 25.040.40

ISBN 2-8318-9485-9

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6–10: Application layer protocol specification – Type 10 elements

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.

3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.

8) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

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

IEC draws attention to the fact that it is claimed that compliance with this standard may involve the use of patents as follows, where the [xx] notation indicates the holder of the patent right:

The following patent rights for Type 10 have been announced by [HI]:

WO publication	Title (WO)
WO 99/046908	Local network, especially ethernet network, with redundancy properties and redundancy manager

The following patent rights for Type 10 have been announced by [SI]:

WO publication	Title (WO)
WO 99/046908	Local network, especially ethernet network, with redundancy properties and redundancy manager
WO 00/026731	Automation system and method for accessing the functionality of hardware components
WO 02/043336	System and method for the parallel transmission of real-time critical and non real-time critical data via switched data networks especially ethernet
WO 02/076033	Synchronous, clocked communication system with local input/output components and method for integrating local input/output components into such a system
WO 03/028258	Method for synchronising nodes of a communication system
WO 03/028259	Communications system and method for synchronising a communications cycle
WO 04/030284	Method for permanent redundant transmission of data telegrams in communication systems
EP 1453230	Synchronisation in einem schaltbaren Datennetz

IEC takes no position concerning the evidence, validity and scope of these patent rights.

The holder of these patent rights has assured the IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the IEC. Information may be obtained from:

[HI]:Hirschmann Automation and Control GmbH
Stuttgarter Straße 45-51
D-72654 Neckartenzlingen
Germany

[SI]:Siemens AG
CT IP L&T
Hr. Hans-Jörg Müller
Otto-Hahn-Ring 6
D-81739 Munich
Germany

Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights other than those identified above. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61158-6-10 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This first edition and its companion parts of the IEC 61158-6 subseries cancel and replace IEC 61158-6:2003. This edition of this part constitutes a technical revision. This part and its Type 10 companion parts also cancel and replace IEC/PAS 62411, published in 2005.

This edition of IEC 61158-6 includes the following significant changes from the previous edition:

- deletion of the former Type 6 fieldbus for lack of market relevance;
- addition of new types of fieldbuses;
- partition of part 6 of the third edition into multiple parts numbered –6-2, –6-3, …

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

FDIS	Report on voting
65C/476/FDIS	65C/487/RVD

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

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

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

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

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

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

INTRODUCTION

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

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6–10: Application layer protocol specification – Type 10 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 10 fieldbus. The term "time-critical" is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard defines in an abstract way the externally visible behavior provided by the Type 10 fieldbus application layer in terms of

a) the abstract syntax defining the application layer protocol data units conveyed between communicating application entities,

b) the transfer syntax defining the application layer protocol data units conveyed between communicating application entities,

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

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

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

a) define the wire-representation of the service primitives defined in IEC 61158-5-10, and

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

This standard specifies the protocol of the Type 10 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 61158-5-10.

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

1.3 Conformance

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

2 Normative references

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

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

IEC 61158–5–10, *Industrial communication networks – Fieldbus specifications – Part 5-10: Application layer service definition – Type 10 elements*

IEC 61784–3–3, *Industrial communication networks – Profiles – Part 3-3: Functional safety fieldbuses – Additional specifications for CPF 3*

ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*

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

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

ISO/IEC 8824, *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 8601, *Data elements and interchange formats – Information interchange – Representation of dates and times*

IEEE 802-2001, *IEEE Standards for local and metropolitan area networks: overview and architecture*

IEEE 802.1AB-2005, *IEEE Standards for Local and Metropolitan Area Networks: Station and Media Access Control Connectivity Discovery*

IEEE 802.1D-2004, *IEEE Standards for local and metropolitan area networks – Media access Control (MAC) Bridges*

IEEE 802.1Q-2005, *IEEE Standards for Local and metropolitan area networks – Virtual Bridged Local Area Networks*

IEEE 802.3-2005, *IEEE Standards for 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*

IEEE 802.11-1999, *IEEE Standards for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*

IEEE 802.15.1-2005, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.1: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)*

IETF RFC 768, *User Datagram Protocol*; available at <http://www.ietf.org>

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

IETF RFC 792, *Internet Control Message Protocol*; available at <http://www.ietf.org>

IETF RFC 826, *An Ethernet Address Resolution Protocol or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*; available at <http://www.ietf.org>

IETF RFC 1034, *Domain names – concepts and facilities*; available at <http://www.ietf.org>

IETF RFC 1112, *Host Extensions for IP Multicasting*; available at <http://www.ietf.org>

IETF RFC 2131, *Dynamic Host Configuration Protocol*; available at <http://www.ietf.org>

IETF RFC 2132, *DHCP Options and BOOTP Vendor Extensions*; available at <http://www.ietf.org>

IETF RFC 2365, *Administratively Scoped IP Multicast*; available at <http://www.ietf.org>

IETF RFC 2474, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*; available at <http://www.ietf.org>

IETF RFC 2674, *Definitions of Managed Objects for Bridges with Traffic Classes, Multicast Filtering and Virtual LAN Extensions*, available at <http://www.ietf.org>

IETF RFC 2737, *Entity MIB (Version 2)*, available at <http://www.ietf.org>

IETF RFC 2863, *The Interfaces Group MIB*, available at <http://www.ietf.org>

IETF RFC 3418, *Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)*, available at <http://www.ietf.org>

IETF RFC 3490, *Internationalizing Domain Names in Applications (IDNA)*; available at <http://www.ietf.org>

IETF RFC 3621, *Power Ethernet MIB*, available at <http://www.ietf.org>

IETF RFC 3636, *Definitions of Managed Objects for IEEE 802.3 Medium Attachment Units (MAUs)*, available at <http://www.ietf.org>

The Open Group — Publication C706, *Technical standard DCE1.1: Remote Procedure Call* (available at <http://www.opengroup.org/onlinepubs/9629399/toc.htm>)

3 Terms, definitions, abbreviations, symbols and conventions

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 8824 terms

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

a) object identifier
b) type

3.1.4 ISO/IEC 9545 terms

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

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

3.2 Additional terms and definitions for distributed automation

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

3.2.1
active connection control object
instance of a certain FAL class that abstracts the interconnection facility (as Consumer and Provider) of an automation device

3.2.2
configuration data base
interconnection information maintained by the ACCO ASE

3.2.3
connection
the logical link between sink and source of attributes and services at different custom interfaces of Custom RT-Auto objects

3.2.4
connection channel
description of a connection between a sink and a source of data items

3.2.5
consumer
node or sink that is receiving data from a producer

3.2.6
consumerID
unambiguous identifier within the scope of the ACCO assigned by the consumer to recognize the internal data of a configured interconnection sink.

3.2.7
data marshaling
the encoding of parameters of the FAL service primitives with respect to their interface definition

NOTE This is part of the abstract ORPC model.

3.2.8
engineering
abstract term that characterizes the client application or device responsible for configuring an automation system via interconnecting data items

3.2.9
event
an instance of a change of conditions

3.2.10
interface
collection of FAL class attributes and services that represents a specific view on the FAL class

3.2.11
interface definition language
syntax and semantic of describing service parameters in a formal way

NOTE This description is the input for the ORPC model, especially for the ORPC wire protocol.

3.2.12
interface pointer
key attribute that unambiguously addresses an object interface instance

3.2.13
logical device
a certain FAL class that abstracts a software component or a firmware component as an autonomous self-contained facility of an automation device

3.2.14
method
<object> a synonym for an operational service which is provided by the server ASE and invoked by a client

3.2.15
object remote procedure call
model for object oriented or component based remote method invocation

3.2.16
physical device
a certain FAL class that abstracts the hardware facilities of an automation device

3.2.17
property
a synonym for ASE attributes which are readable or writeable via operational ASE services

NOTE These services are generally named “get_<Attribute Name>” or “set_<Attribute Name>” and correspond with the IDL keywords “propget” and “propput”.

3.2.18
provider
source of a data connection.

3.2.19
providerID
an unambiguous identifier within the scope of the ACCO assigned by the provider to recognize the internal data of a configured connection source

3.2.20
quality code
additional status information of a data item

3.2.21
quality code aware
attribute of the RT-Auto class that indicates that an RT-Auto object uses a status code for its data items

3.2.22
quality code unaware
opposite of quality code aware

3.2.23
RT-Auto
an FAL class that abstracts the automation function as a process-related component of an automation device

3.2.24
runtime object model
objects that exist in a device together with their interfaces and methods that are accessible

3.3 Additional terms and definitions for decentralized periphery

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

3.3.1
alarm
activation of an event that shows a critical state

3.3.2
alarm ack
acknowledgment of an event that shows a critical state

3.3.3
alarm data object
object(s) which represent critical states referenced by device/slot/subslot/alarm type

3.3.4
allocate
take a resource from a common area and assign that resource for the exclusive use of a specific entity

3.3.5
application
function or data structure for which data is consumed or produced

3.3.6
application layer interoperability
capability of application entities to perform coordinated and cooperative operations using the services of the FAL

3.3.7
application objects
multiple object classes that manage and provide a run time exchange of PDUs across the network and within the network device

3.3.8
application process
part of a distributed application on a network, which is located on one device and unambiguously addressed

3.3.9
application process identifier
distinguishes multiple application processes used in a device

NOTE Application process identifier is assigned by PROFIBUS International (PI).

3.3.10
application process object
component of an application process that is identifiable and accessible through an FAL application relationship

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

3.3.11
application process object class
a class of application process objects defined in terms of the set of their network-accessible attributes and services

3.3.12
application relationship
cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation. This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities

3.3.13
application relationship application service element
application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.3.14
application relationship endpoint
context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship

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

3.3.15
attribute
description of an externally visible characteristic or feature of an object

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

3.3.16
backup
status of the IO AR, which indicates that it, is in the standby state

3.3.17
behavior
indication of how an object responds to particular events

3.3.18
channel
representation of a single physical or logical link of an input or output application object of a server to the process in order to support addressing of diagnosis information

NOTE The channel typically represents a single connector or clamp as a real interface of a module or sub-module. This reference is used to identify points of failure within diagnosis PDUs.

3.3.19
channel related diagnosis
information concerning a specific element of an input or output application object, provided for maintenance purposes

EXAMPLE open loop

3.3.20
class
a set of objects, all of which represent the same kind of system component

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

3.3.21
class attributes
attribute that is shared by all objects within the same class

3.3.22
class code
unique identifier assigned to each object class

3.3.23
class specific service
service defined by a particular object class to perform a required function which is not performed by a common service

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

3.3.24
clear
status of the IO controller, which indicates that the control algorithm is currently not running

3.3.25
client

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

b) initiator of a PDU to which a server reacts

3.3.26
common profile
a collection of device independent information and functionality providing consistency between all devices

3.3.27
communication data object
object(s) which are parameter of communication relationships and referenced by device/ slot/ subslot/ index

3.3.28
configuration check
comparison of the expected IO-Data object structuring of the client with the real IO-Data object structuring to the server in the start-up phase

3.3.29
configuration fault
an unacceptable difference between the expected IO-Data object structuring and the real IO-Data object structuring, as detected by the server

3.3.30
configuration identifier
representation of a portion of IO Data of a single input- and/or output-module of a server

3.3.31
consume
act of receiving data from a provider

3.3.32
consumer
node or sink receiving data from a provider

3.3.33
context management
network-accessible information (communication objects) that supports managing the operation of the fieldbus system, including the application layer

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

3.3.34
conveyance path
unidirectional flow of APDUs across an application relationship

3.3.35
cyclic
repetitive in a regular manner

3.3.36
data consistency
means for coherent transmission and access of the input- or output-data object between and within client and server

3.3.37
device
physical hardware connected to the link

NOTE A device may contain more than one node.

3.3.38
device ID
a vendor assigned device type identification

3.3.39
device profile
a collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.3.40
diagnosis data object
object(s) which contains diagnosis information referenced by device/slot/subslot/index

3.3.41
diagnosis information
all data available at the server for maintenance purposes

3.3.42
dynamic reconfiguration
change of IO data objects without interruption of an established application relationship and continuous updating of non-changed IO data objects

3.3.43
endpoint
one of the communicating entities involved in a connection

3.3.44
engineering
abstract term that characterizes the client application or device responsible for configuring an automation system

3.3.45
error
discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.3.46
error class
general grouping for related error definitions and corresponding error codes

3.3.47
error code
identification of a specific type of error within an error class

3.3.48
event
instance of a change of conditions

3.3.49
extended channel related diagnosis
information concerning a specific element of a specific application object, provided for maintenance purposes

EXAMPLE Link Fail

3.3.50
frame
unit of data transfer on a link

3.3.51
identification data object
object(s) that contain information about device, module and sub-module manufacturer and type referenced by device/slot/subslot/index

3.3.52
implicit AR endpoint
AR endpoint that is defined locally within a device without use of the create service

3.3.53
index
address of a record data object within an application process

3.3.54
instance
the actual physical occurrence of an object within a class that identifies one of many objects within the same object class

3.3.55
instance attributes
attribute that is unique to an object instance and not shared by the object class

3.3.56
instantiated
object that has been created in a device

3.3.57
invocation
act of using a service or other resource of an application process

NOTE Each invocation represents a separate thread of control that may be described by its context. Once the service completes, or use of the resource is released, the invocation ceases to exist. For service invocations, a service that has been initiated but not yet completed is referred to as an outstanding service invocation. Also for service invocations, an Invoke ID may be used to unambiguously identify the service invocation and differentiate it from other outstanding service invocations.

3.3.58
IO controller
controlling device, which acts as client for several IO devices (field devices)

NOTE This is usually a programmable controller or a distributed control system.

3.3.59
IO data object
object designated to be transferred cyclically for the purpose of processing and referenced by device/slot/subslot

3.3.60
IO device
field device which acts as server for IO operation

3.3.61
IO parameter server
server for application parameter of IO devices (client)

NOTE This is usually a device to backup parameter data and to log online changes of device parameter.

3.3.62
IO subsystem
subsystem composed of one IO controller and all its associated IO devices

3.3.63
IO supervisor
engineering device which manages commissioning and diagnosis of an IO system

3.3.64
IO system
system composed of all its IO subsystems

NOTE As an example a PLC with more than one IO controller (network interface) controls one IO system composed of an IO subsystems for each IO controller.

3.3.65
Isochronous mode
IO system operating tightly synchronized with a jitter of less than 1 µs

3.3.66
member
piece of an attribute that is structured as an element of an array

3.3.67
message
synonym for frame

3.3.68
method
a synonym for an operational service which is provided by the server ASE and invoked by a client

3.3.69
module
hardware or logical component of a physical device

3.3.70
network
a set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.3.71
object
abstract representation of a particular component within a device, usually a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behavior

3.3.72
object specific service
service unique to the object class which defines it

3.3.73
operate
status of the IO controller that indicates that the control algorithm is currently running

3.3.74
packet
frame

3.3.75
peer
role of an AR endpoint in which it is capable of acting as both client and server

3.3.76
physical device
automation or other network device

3.3.77
point-to-point connection
connection that exists between exactly two application objects

3.3.78
primary
status of the IO AR that indicates that it is in the operating state

NOTE Besides a primary IO AR a backup IO AR may exist. In example used for redundancy and dynamic reconfiguration of IO data.

3.3.79
provider
node or source sending data to one or many consumer

3.3.80
PTCP domain
certain number of PTCP subdomains in one IP subnet

3.3.81
PTCP subdomain
certain amount of DTEs with synchronized clocks

3.3.82
record data object
object(s) which are already pre-processed and transferred acyclically for the purpose of information or further processing and referenced by device/slot/subslot/index

3.3.83
resource
processing or information capability

3.3.84
run
status of the IO controller which indicates that the control algorithm is currently operating

3.3.85
server

a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request

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

3.3.86
service
operation or function than an object and/or object class performs upon request from another object and/or object class

3.3.87
slot
address of a structural unit within an IO device

NOTE Within a modular device, a slot typically addresses a physical module. Within compact devices, a slot typically addresses a logical function or virtual module.

3.3.88
stop
status of the IO controller which indicates that the control algorithm is currently not running

3.3.89
submodule
hardware or logical component of a module

3.3.90
subslot
address of a structural unit within a slot

NOTE A subslot may address a physical interface for submodules within a module. Generally, a subslot is a second level to structure data within a device.

3.3.91
vendor ID
central administrative number used as manufacturer identification

NOTE The vendor ID is assigned by PROFIBUS International (PI).

3.4 Additional abbreviations and symbols for distributed automation

ACCO	Active Connection Control Object
IDL	Interface Definition Language
IP	Internet Protocol
DNS	Domain Name Service
LDev	Logical Device
ORPC	Object Remote Procedure Call
PDev	Physical Device
QoS	Quality of Service
QC	Quality Code
RT	Runtime
RT-Auto	Runtime Automation Object
TCP	Transmission Control Protocol

3.5 Additional abbreviations and symbols for decentralized periphery

AE	Application Entity
AL	Application Layer
ALME	Application Layer Management Entity
ALP	Application Layer Protocol
ALPMI	Alarm Protocol Machine Initiator
ALPMR	Alarm Protocol Machine Responder
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Identifier
APO	Application Object
AR	Application Relationship
AREP	Application Relationship End Point
ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
BMC	Best Master Clock
CM	Context Management
Cnf	Confirmation
CR	Communication Relationship
CREP	Communication Relationship End Point
DCE	OSF Distributed Computing Environment
DCP	Discovery and basic Configuration Protocol
DCPMCR	DCP Multicast Receiver
DCPMCS	DCP Multicast Sender
DCPUCR	DCP Unicast Receiver
DCPUCS	DCP Unicast Sender
DHCP	Dynamic Host Configuration Protocol
DIM	Device Interface Module
DL-	(as a prefix) Data Link-
DLC	Data Link Connection
DLL	Data Link Layer
DLPDU	Data Link-Protocol Data Unit
DLSDU	DL-service-data-unit
DNS	Domain Name Service
DTE	Date Terminal Equipment
FAL	Fieldbus Application Layer
FIFO	First In First Out
GSDML	Generic Station Description Markup Language
I&M	Identification and Maintenance Profile
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
ID	Identifier
IEC	International Electrotechnical Commission
IFW	RT_CLASS_3 Forwarding Protocol Machine
Ind	Indication
IOCS	Input Output Object Consumer Status
IOPS	Input Output Object Provider Status
IP	Internet Protocol
IR	Isochronous Relay
IRT	Isochronous Real Time Protocol
ISO	International Organization for Standardization
IsoM	Isochronous Mode
LED	Light Emitting Diode
LLDP	Link Layer Discovery Protocol
LME	Layer Management Entity
lsb	Least Significant Bit
LT	Length/Type
MAC	Medium Access Control
msb	Most Significant Bit
NCA	Network Computing Architecture

OSI	Open Systems Interconnect
PDU	Protocol Data Unit
PI	PROFIBUS International see <http://www.profinet.com>
PL	Physical Layer
PTCP	Precision Transparent Clock Protocol
QoS	Quality of Service
Req	Request
RPC	Remote Procedure Call
Rsp	Response
RT	Real Time Protocol
RTA	Real Time Protocol Acyclic
RTC	Real Time Protocol Cyclic
RTE	Real Time Ethernet
SDU	Service Data Unit
TLV	Type Length Value (coding rule)
UDP	User Datagram Protocol
UUID	Universal Unique Identifier
VLAN	Virtual Local Area Network

3.6 Additional abbreviations and symbols for media redundancy

MRC	Media Redundancy Client
MRM	Media Redundancy Master
MRP	Media Redundancy Protocol

3.7 Conventions

3.7.1 General concept

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate clause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158–5–10. 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 61158–5–10 standard. The service specification defines the services that are provided by the ASE.

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

3.7.2 Conventions for distributed automation

3.7.2.1 IDL – Interface Definition Language

3.7.2.1.1 Basic information

The Interface Definition Language (IDL) is a language for specifying interfaces, operations (procedures or functions), parameters to these operations, and data types.

Notation

The IDL is a case-sensitive language. The syntax of IDL is described using an extended BNF (Backus-Naur Form) notation. The meaning of the BNF notation is as follows:

- Brackets ([ ]) enclose an optional part of the syntax.
- Ellipsis points (...) indicate that the left clause can be repeated either zero or more times if it is optional or one or more times if it is required. The vertical bar (|) indicates alternative productions; it is read as "or."
- Language punctuation that does not conflict with punctuation characters used in the BNF notation appears in a production in the appropriate position. Language punctuation that does conflict with punctuation characters used in the BNF notation is enclosed in less-than and greater-than symbols; for example, <[>. Note particularly that when ' (single quotation) or " (double quotation) appear in a production, they are a part of the language and shall appear in IDL source.

Elements in the grammar that are capitalized are terminals of the grammar. For example, <Identifier> is not further expanded. Also, keywords of the language are terminals of the grammar. For example, the keyword boolean is not further expanded.

3.7.2.1.2 Keywords and reserved words

Some keywords are reserved words, and shall not be used as identifiers. Keywords that are not reserved may be used as identifiers, except when used as attributes (that is, within [](brackets)). Reserved words are keywords of the language that shall not be used as user identifiers.

IDL reserved words are given in the following list:

boolean	byte	case	char
const	default	double	enum
FALSE	float	handle_t	hyper
import	int	interface	long
NULL	pipe	short	small
struct	switch	TRUE	typedef
union	unsigned	void	

The following list gives the IDL keywords that are reserved when in the context of an attribute: that is, between [](brackets):

align	broadcast	broadcast	comm_status
context_handle	endpoint	first_is	handle
idempotent	ignore	implicit_handle	in
last_is	length_is	local	max_is
maybe	out	ptr	ref
size_is	string	switch_is	switch_type
transmit_as	uuid	version	

3.7.2.1.3 Identifiers

Each object is named with an unique identifier. The maximum length of an identifier is 31 characters.

Some identifiers are used as a base from which the compiler constructs other identifiers. These identifiers have further restrictions on their length. The character set for identifiers contains the alphabetic characters A to Z and a to z, the digits 0 to 9, and the _ (underbar) character. An identifier shall start with an alphabetic character or the _ (underbar) character.

3.7.2.1.4 IDL punctuation

The punctuation used in IDL consists of the following characters:

- The **.** (dot)
- The **,** (comma)
- The pair ()(parentheses)
- The pair [](brackets)
- The pair {}(braces)
- The **;** (semicolon)
- The **:** (colon)
- The ***** (asterisk)
- The **’** (single quote)
- The **"** (double quote)
- The **=** (equal sign)

3.7.2.1.5 Interface definition structure

An interface definition written in IDL has the following structure:

```
<interface>::=<interface_header>{<interface_body>}
```

3.7.2.1.6 Interface header

The structure of the interface header is as follows:

```
<interface_header>::=<[><interface_attributes><]>interface <Identifier>
```

where:

```
<interface_attributes>::=<interface_attribute>[,<interface_attribute>]...
<interface_attribute>::=uuid (<Uuid_rep>)
   |version (<Integer_literal>[,<Integer_literal>])
   |endpoint (<port_spec>[,<port_spec>]...)
   |local
   |pointer_default (<ptr_attr>)
<port_spec>::=<Family_string>:<[><Port_string><]>
```

If an interface defines any operations, exactly one of the uuid attribute or the local attribute shall be specified. Whichever is specified shall appear exactly once.

It is permissible to have neither the uuid nor the local attribute if the interface defines no operations. The version attribute may occur at most once.

3.7.2.1.7 Interface body

The structure of the interface body is as follows:

```
<interface_body>::=[<import>...]<interface_component>
[<interface_component>...]
```

where:

```
<import>::=import <import_list>;
<interface_component>::=<export>
   |<op_declarator>;
<export>::=<type_declarator>;
   |<const_declarator>;
   |<tagged_declarator>;
```

3.7.2.1.8 Further possibilities

It is possible to use comments. The IDL allows to declare constants. Type declarations are subdivided into base types and the fundamental integer, char, boolean, byte, void, handle_t types, constructed types, structures, unions, enumerated types, pipes and arrays.

3.7.3 Conventions for decentralized periphery

3.7.3.1 Abstract syntax conventions

3.7.3.1.1 PDUs are described as octets or groups of octets

a) Groups of octets separated by a comma appear in the order they are transferred If optional octets are not present the following octets appear without a gap

b) If octets or groups of octets are grouped within "{ }" the order is arbitrary

c) If octets or groups of octets are marked with "*" they may appear more than once. If it is used within a "{ }" section they may appear mixed with other octets or group of octets of this section.

d) Octets can be grouped or values can be assigned within "( )"

e) If octets or groups of octets are grouped within "[ ]" the group can be omitted

f) Complex APDUs may be built by means of substitutions (sub-structures)

g) Exclusive selections of octets or groups of octets are separated by "^"

NOTE 1 The formal PDU example

AP_PDU = Octet1, OctetGroup1, [Octet2], [Octet3], {[OctGroup2*], OctetGroup3 ^ Octet4}

According to this the following variants are valid on the wire (non exhaustive):

Variant 1: Octet1, OctetGroup1, Octet2, Octet3, OctetGroup2, OctetGroup3

Variant 2: Octet1, OctetGroup1, Octet2, Octet3, OctetGroup2, OctetGroup2, OctetGroup2, OctetGroup3

Variant 3: Octet1, OctetGroup1, OctetGroup2, OctetGroup2, OctetGroup2, OctetGroup3, OctetGroup2

Variant 4: Octet1, OctetGroup1, OctetGroup2, OctetGroup3, OctetGroup2, OctetGroup2, OctetGroup2, OctetGroup2

Variant 5: Octet1, OctetGroup1, Octet3, Octet4

NOTE 2 The arbitrary order implies that groups of octets are characterised by a special header that is described within the coding rules.

NOTE 3 The APDU syntax for RTA-, and RTC-PDU implies that according to the maximum DLSDU an APDU does not exceed 1 440 octets in total.

NOTE 4 The APDU syntax for CL RPC implies that an IO controller supports a minimal ASDU size of 4 096 octets in total and does not exceed 2^{32}-64 octets in total. The minimal ASDU size is derived from the expected size of configuration, parameter and diagnosis data of an enhanced IO device.

3.7.3.2 Convention for the encoding of reserved bits and octets

The term "reserved" may be used to describe bits in octets or whole octets. All bits or octets that are reserved should be set to zero at the sending side and shall not be tested at the receiving side except it is explicitly stated or if the reserved bits or octets are checked by a state machine.

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

3.7.3.3 Conventions for the common codings of specific field octets

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

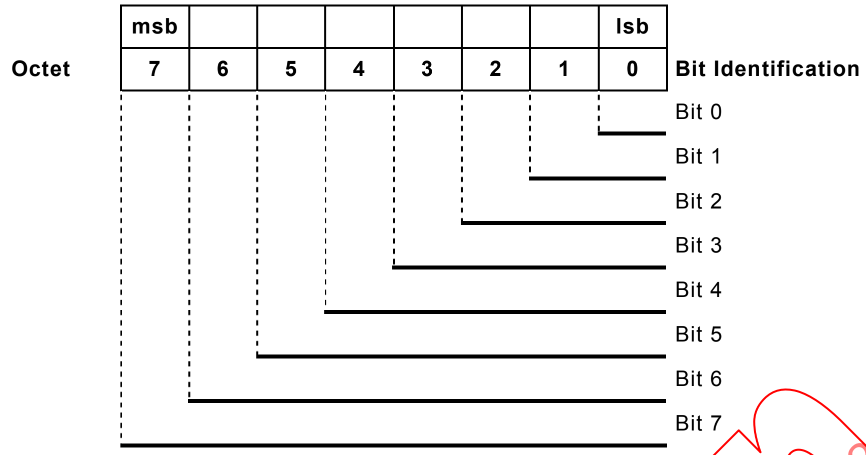

Figure 1 – Common structure of specific fields

Several bit may be grouped as group of bit. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. Alias names may be used for each bit or group of bits or they may be marked as reserved. The grouping of individual bits shall be in ascending order without gaps. The values for a group of bits may be represented as binary, decimal or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 8 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest number of the group represents the msb concerning the grouped bit.

EXAMPLE 1 Description and relation for the specific field octet

Bit 0: reserved.

Bit 1-3: Reason_Code The decimal value 2 for the Reason_Code means general error.

Bit 4-7: shall always set to one.

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

(msb) Bit 7 = 1,

Bit 6 = 1,

Bit 5 = 1,

Bit 4 = 1,

Bit 3 = 0,

Bit 2 = 1,

Bit 1 = 0,

(lsb) Bit 0 = 0.

The bit combination "0-1-0" for Bit 1-3 equals the decimal value 2.

3.7.3.4 Conventions for the common codings of specific field consisting of two subsequent octets

APDUs may contain specific fields that carry information in a primitive and condensed way. These fields shall be coded in the order according to Figure 2 and Figure 3.

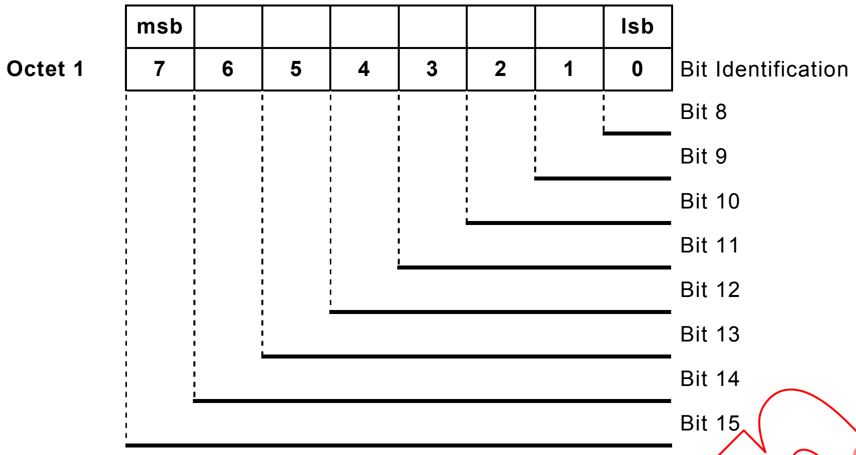

Figure 2 – Common structure of specific fields for octet 1 (high)

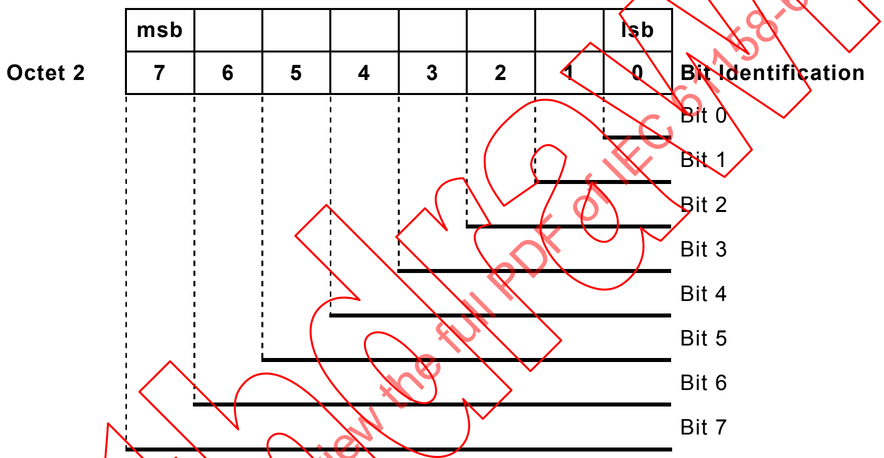

Figure 3 – Common structure of specific fields for octet 2 (low)

Several bits may be grouped as group of bit. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. Alias names may be used for each bit or group of bits or they may be marked as reserved. The grouping of individual bits shall be in ascending order without gaps. The values for a group of bits may be represented as binary, decimal or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 16 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest number of the group represents the msb concerning the grouped bit.

3.7.3.5 Conventions for the common coding of specific field consisting of four subsequent octets

APDUs may contain specific fields that carry information in a primitive and condensed way. These fields shall be coded in the order according to Figure 4, Figure 5, Figure 6 and Figure 7.

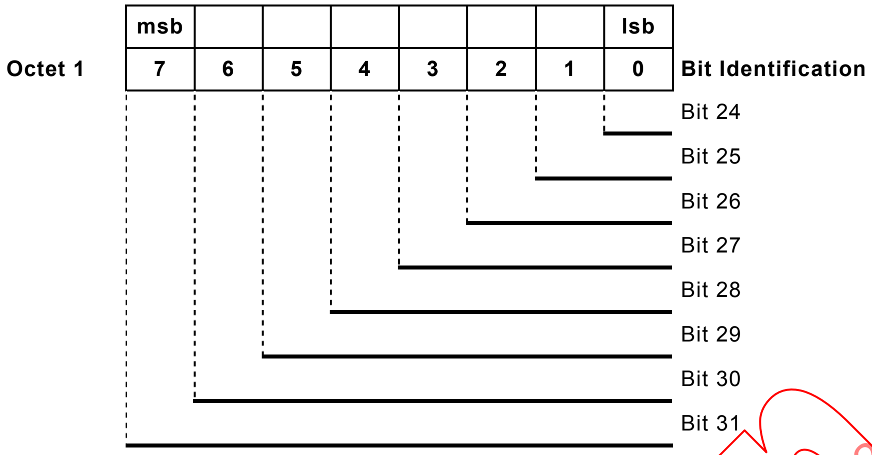

Figure 4 – Common structure of specific fields for octet 1 (high)

Figure 5 – Common structure of specific fields for octet 2

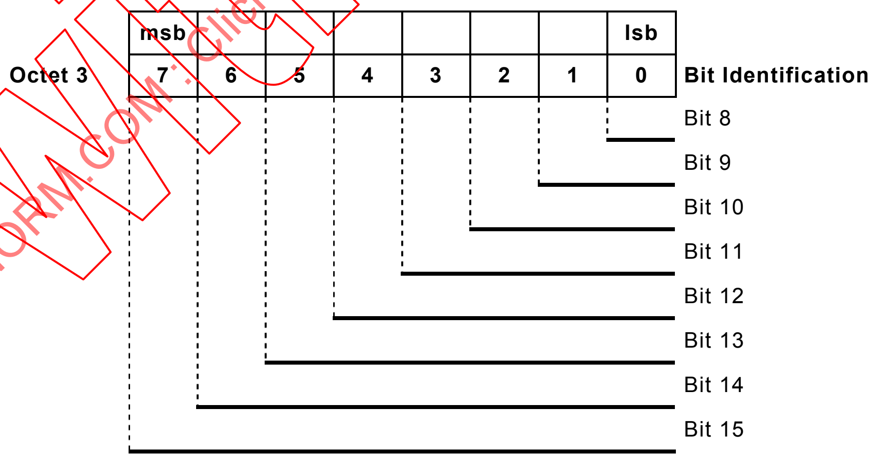

Figure 6 – Common structure of specific fields for octet 3

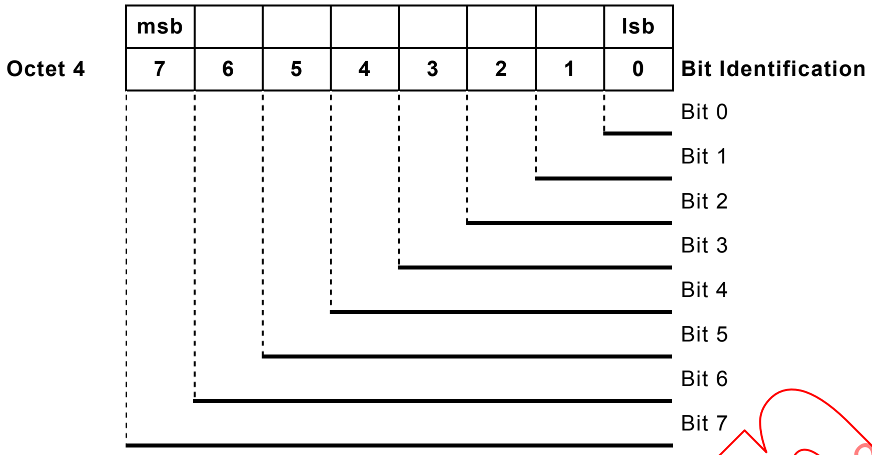

Figure 7 – Common structure of specific fields for octet 4 (low)

Bits may be grouped as group of bits. Each bit or group of bits shall be addressed by its Bit Identification (e.g. Bit 0, Bit 1 to 4). The position within the octet shall be according to the figure above. Alias names may be used for each bit or group of bits or they may be marked as reserved. The grouping of individual bits shall be in ascending order without gaps. The values for a group of bits may be represented as binary, decimal or hexadecimal values. This value shall only be valid for the grouped bits and can only represent the whole octet if all 32 bits are grouped. Decimal or hexadecimal values shall be transferred in binary values so that the bit with the highest number of the group represents the msb concerning the grouped bits.

3.8 Conventions used in state machines

The protocol sequences are described by means of State Machines.

In state diagrams states are represented as boxes state transitions are shown as arrows. Names of states and transitions of the state diagram correspond to the names in the textual listing of the state transitions.

The textual listing of the state transitions is structured as follows, see also Table 1.

- The first row contains the name of the transition.
- The second row defines the current state.
- The third row contains an optional event followed by Conditions starting with a “/” as first line character and finally followed by the Actions starting with a “=>” as first line character.
- The last row contains the next state.

If the event occurs and the conditions are fulfilled the transition fires, e.g. the actions are executed and the next state is entered.

The layout of a Machine description is shown in Table 1. The meaning of the elements of a State Machine Description are shown in Table 2.

Table 1 – State machine description elements

#	Current state	Event or condition => action	Next state

Table 2 – Description of state machine elements

Description element	Meaning
Current state Next state	Name of the given states
#	Name or number of the state transition
Event	Name or description of the event
/Condition	Boolean expression. The preceding "\" is not part of the condition
=> Action	List of assignments and service or function invocations. The preceding "=>" is not part of the action

The conventions used in the state machines are shown in Table 3.

Table 3 – Conventions used in state machines

Convention	Meaning
:=	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.
>>	A semantic condition with the meaning "newer"
<<	A semantic condition with the meaning "older"
&&	Logical "AND"
||	Logical "OR"

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

If (condition)
actions
else
actions
	This construct allows the execution of alternative actions depending on some condition (which might be the value of some identifier or the outcome of a previous action) within one transition.
	The parts beginning with "else" can be ommitted if there is no action if the condition is not fulfilled

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

In addition the following description elements are used:

Wildcard in names
name_XXX: "XXX" is used as wildcard string for all names beginning with "name".

The typical use of a wildcard is an Event. In this context there are as many state transitions as possible events for this wildcard exists.

Conditional Macro
<CONDITIONAL -MACRO-NAME>

<

CONDITION1: macrobody1

CONDITION2: macrobody2

...

>

CONDITION1, CONDITION2, etc. define all possible cases for the conditional macro. The “CONDITIONAL-MACRO-NAME” acts as place holder for the “macrocode” depending on the result of the condition.

Replacement Macro
<REPLACEMENT-MACRO-NAME>

<

XXX= Name1 : macrobody1

XXX= Name2 : macrobody2

...

>

Name1, Name2, etc. define all possible cases for the replacement macro. The “REPLACEMENT-MACRO-NAME” acts as place holder for the “macrocode” depending on the value of the current use of the wildcard XXX.

EXAMPLE

<SERVICE_REQ_PARA>

<

XXX=Read: Para1, Para2

XXX=Write: Para1, Para2, Para3

>

when called in

MSAB_XXX.req(<SERVICE_REQ_PARA>)

will result in

MSAB_Read.req(Para1, Para2) or MSAB_Write.req(Para1, Para2, Para3)

4 Application layer protocol specification for common protocols

4.1 FAL syntax description

4.1.1 DLPDU abstract syntax reference

4.1.1.1 General

Table 4, Table 5, and Table 6 give an outline of the abstract syntax of the DLPDU according to IEEE 802.3, IEEE 802.11 and IEEE 802.15.1.

4.1.1.2 IEEE 802.3

The encoding and decoding of the fields in Table 4 shall be according to IEEE 802.3 for the DLPDU.

Table 4 – IEEE 802.3 DLPDU syntax

DLPDU name	DLPDU structure
DLPDU	Preamble [a], StartFrameDelimiter, DestinationAddress, SourceAddress, DLSDU [b], DLPDU_Padding* [c], FrameCheckSequence
DLSDU	[VLAN] [d], LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ MRP-PDU ^ CL-RPC-PDU ^ LLDP-PDU [e] ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU [e] ^ MRRT-PDU
VLAN	LT(=0x8100), TagControlInformation

NOTE 1 According to IEEE 802.3 the DLPDUs have a minimum length of 64 octets (excluded Preamble, Start Frame Delimiter).

NOTE 2 For the IEEE 802.3 frames with VLAN tag is the minimum frame size increased to 68 octets in order to guaranty the minimum frame size of 64 octets after removing the VLAN tag by a bridge.

[a] The field contains at least 7 octets

[b] The minimum DLSDU size is 2 octets.

[c] The number of padding octets shall be in the range of 0..46 depending on the DLSDU size. The value shall be set to zero.

[d] The VLAN field can be omitted in case of optimized RT transportation. The field VLAN may be set by the encoder but it may be discarded by intermediate bridges. The decoder shall accept DLPDUs with or without VLAN fields. The VLAN field shall be omitted for RT_CLASS_3 PDUs.

[e] The VLAN field should be omitted.

4.1.1.3 IEEE 802.11

The encoding and decoding of the fields in Table 5 shall be according to IEEE 802.11 for the DLPDU.

Table 5 – IEEE 802.11 DLPDU syntax

DLPDU name	DLPDU structure
DLPDU	DLPDU_1 ^ DLPDU_2 ^ DLPDU_3 ^ DLPDU_4
DLPDU_1	FrameControl, DestinationAddress, SourceAddress, BSSID, SequenceControl, [QoSControl] [b], DLSDU, DLPDU_Padding* [a], FrameCheckSequence
DLPDU_2	FrameControl, DestinationAddress, BSSID, SourceAddress, SequenceControl, [QoSControl] [b], DLSDU, DLPDU_Padding* [a], FrameCheckSequence
DLPDU_3	FrameControl, BSSID, SourceAddress, DestinationAddress, SequenceControl, [QoSControl] [b], DLSDU, DLPDU_Padding* [a], FrameCheckSequence
DLPDU_4	FrameControl, ReceiverAddress, TransmitterAddress, DestinationAddress, SequenceControl, SourceAddress, [QoSControl] [b], DLSDU, DLPDU_Padding* [a], FrameCheckSequence
DLSDU	LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ MRP-PDU ^ CL-RPC-PDU ^ LLDP-PDU ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU ^ MRRT-PDU

NOTE 1 For definition of FrameControl see IEEE 802.11.

NOTE 2 For definition of BSSID see IEEE 802.11.

NOTE 3 For definition of SequenceControl see IEEE 802.11.

NOTE 4 For definition of QoSControl see IEEE 802.11.

NOTE 5 For definition of ReceiverAddress see IEEE 802.11.

NOTE 6 For definition of TransmitterAddress see IEEE 802.11.

[a] The number of padding octets shall be in the range of 0..46 depending on the DLSDU size. The value shall be set to zero.

[b] This field should exist for UDP-RTC-PDU and UDP-RTA-PDU and shall exist for RTC-PDU and RTA-PDU.

4.1.1.4 IEEE 802.15.1

The encoding and decoding of the fields in Table 6 shall be according to IEEE 802.15.1 for the DLPDU.

Table 6 – IEEE 802.15.1 DLPDU syntax

DLPDU name	DLPDU structure
DLPDUHEADER	Access Code, Packet Header, Payload Header, L2CAP Header, BNEPType(0)
DLPDU	DLPDUHEADER, DestinationAddress, SourceAddress, DLSDU [a], FrameCheckSequence
DLSDU	[VLAN] [b], LT, SAPDU ^ FIDAPDU
SAPDU	UDP-RTC-PDU ^ UDP-RTA-PDU ^ MRP-PDU ^ CL-RPC-PDU ^ LLDP-PDU [c] ^ ICMP-PDU
FIDAPDU	FrameID, RTC-PDU ^ RTA-PDU ^ DCP-PDU ^ PTCP-PDU [c] ^ MRRT-PDU
VLAN	LT(=0x8100), TagControlInformation

NOTE 1 For definition of Access Code see IEEE 802.15.1-2005, 8.6.3.

NOTE 2 For definition of Packet Header see IEEE 802.15.1-2005, 8.6.4.

NOTE 3 For definition of Payload Header see IEEE 802.15.1-2005, 8.6.6.2.

NOTE 4 For definition of L2CAP Header see IEEE 802.15.1-2005, 14.3.1.

NOTE 5 For definition of BNEPType see IEEE 802.15.1-2005.

DLPDU name	DLPDU structure
[a] The minimum DLSDU size is 2 octets. [b] This field should exist for UDP-RTC-PDU and UDP-RTA-PDU and shall exist for RTC-PDU and RTA-PDU. [c] The VLAN field should be omitted.	

4.1.2 Data types

4.1.2.1 Notation for the Boolean type

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

4.1.2.2 Notation for the Integer type

Integer8 ::= INTEGER (-128..+127) -- range $-2^{7} <= I <= 2^{7}-1$

Integer16 ::= INTEGER (-32 768..+32 767) -- range $-2^{15} <= I <= 2^{15}-1$

Integer32 ::= INTEGER -- range $-2^{31} <= I <= 2^{31}-1$

Integer64 ::= INTEGER -- range $-2^{63} <= I <= 2^{63}-1$

4.1.2.3 Notation for the Unsigned type

Unsigned8 ::= INTEGER (0..255) -- range $0 <= I <= 2^{8}-1$

Unsigned16 ::= INTEGER (0..65 535) -- range $0 <= I <= 2^{16}-1$

Unsigned32 ::= INTEGER -- range $0 <= I <= 2^{32}-1$

Unsigned64 ::= INTEGER -- range $0 <= I <= 2^{64}-1$

4.1.2.4 Notation for the Floating Point type

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

Floating64 ::= BIT STRING SIZE (8) -- IEC 60559 Double precision

4.1.2.5 Notation for the OctetString type

OctetString ::= OCTET STRING -- For generic use

4.1.2.6 Notation for VisibleString type

VisibleString ::= VISIBLE STRING -- IEC 646 – International Reference Version without the “del”(coding 0x7F) character

4.1.2.7 Notation for BinaryDate type

BinaryDate ::= OctetString SIZE (7)

4.1.2.8 Notation for TimeOfDay type

TimeOfDay with date indication ::= OctetString SIZE (6)

TimeOfDay without date indication ::= OctetString SIZE (4)

4.1.2.9 Notation for TimeDifference type

TimeDifference with date indication ::= OctetString SIZE (6)

TimeDifference without date indication ::= OctetString SIZE (4)

4.1.2.10 Notation for Network Time type

Network Time ::= OctetString SIZE (8)

4.1.2.11 Notation for Network Time Difference type

Network Time Difference ::= OctetString SIZE (8)

4.2 Transfer syntax

4.2.1 Coding of basic data types

4.2.1.1 General Encoding

– The encoding of values shall be big endian if not explicitly otherwise stated.

4.2.1.2 Encoding of a Boolean value

- The encoding of a Boolean value shall be primitive. The ContentsOctets shall consist of a single octet.
- If the Boolean value is FALSE, the ContentsOctets shall be 0 (zero). If the Boolean value is TRUE, the ContentsOctets shall be 0xff.

4.2.1.3 Encoding of an Integer value

- The encoding of a fixed-length Integer value of Integer8, Integer16, Integer32, and Integer64 types shall be primitive, and the ContentsOctets shall consist of exactly one, two, four, or eight octets, respectively.
- The ContentsOctets shall be a two's complement binary number equal to the integer value, and consist of bits 7 to 0 of the first octet, followed by bits 7 to 0 of the second octet, followed by bits 7 to 0 of each octet in turn up to and including the last octet of the ContentsOctets.

NOTE The value of a two's complement binary number is derived by numbering the bits in the ContentsOctets, starting with bit 0 of the last octet as bit zero and ending the numbering with bit 7 of the first octet. Each bit is assigned a numerical value of 2^N, where N is its position in the above numbering sequence. The value of the two's complement binary number is obtained by adding the numerical values assigned to each bit for those bits which are set to one, excluding bit 7 of the first octet, and then reducing this value by the numerical value assigned to bit 7 of the first octet if that bit is set to one.

4.2.1.4 Encoding of an Unsigned value

- The encoding of a fixed-length Unsigned value of Unsigned8, Unsigned16, Unsigned32, and Unsigned64 types shall be primitive, and the ContentsOctets shall consist of exactly one, two, four, or eight octets, respectively.
- The ContentsOctets shall be a binary number equal to the Unsigned value, and consist of bits 7 to 0 of the first octet, followed by bits 7 to 0 of the second octet, followed by bits 7 to 0 of each octet in turn up to and including the last octet of the ContentsOctets.

NOTE The value of a binary number is derived by numbering the bits in the ContentsOctets, starting with bit 0 of the last octet as bit zero and ending the numbering with bit 7 of the first octet. Each bit is assigned a numerical value of 2^N, where N is its position in the above numbering sequence. The value of the binary number is obtained by adding the numerical values assigned to each bit for those bits which are set to one.

4.2.1.5 Encoding of a Floating-Point value

- The encoding of a fixed-length Floating-Point value of Floating32 and Floating64 types shall be primitive, and the ContentsOctets shall consist of exactly four, or eight octets, respectively.
- The ContentsOctets shall contain floating-point values defined in conformance with IEC 60559. The sign is encoded in bit 7 of the first octet. It is followed by the exponent starting from bit 6 of the first octet, and then the mantissa starting from bit 6 of the second octet for Floating32 or Floating64.

4.2.1.6 Encoding of a Visible String value

- The encoding of a variable length VisibleString value shall be primitive.
- There is no Length field; the length is encoded implicitly.
- The ContentsOctets shall be a sequence of octets. The leftmost string element is encoded in the first octet, followed by the second octet, followed by each octet in turn up to and including the last octet as rightmost of the ContentsOctets.

4.2.1.7 Encoding of an OctetString value

- The encoding of a variable length OctetString value shall be primitive.
- There is no Length field; the length is encoded implicitly.
- The ContentsOctets shall be a sequence of octets. The leftmost string element is encoded in the first octet, followed by second octet, followed by each octet in turn up to and including the last octet as rightmost of the ContentsOctets.

4.2.1.8 Encoding of a BinaryDate value

- The encoding of a BinaryDate value shall be primitive.
- The Length field shall indicate as a binary number the number of octets in the BinaryDate value.
- The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 8:

bits octets	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	0…59 999 ms
2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
3	RSV	RSV	2^5	2^4	2^3	2^2	2^1	2^0	0…59 min
4	SU	RSV	RSV	2^4	2^3	2^2	2^1	2^0	0…23 hours
	day of week			day of month					1…7 day of week
5	2^2	2^1	2^0	2^4	2^3	2^2	2^1	2^0	1…31 day of month
6	RSV	RSV	2^5	2^4	2^3	2^2	2^1	2^0	1…12 months
7	RSV	2^6	2^5	2^4	2^3	2^2	2^1	2^0	0 … 50 years 2000 to 2050 51 … 99 years 1951 to 1999
	msb							lsb	

NOTE To avoid the Y2K problem the values of the year are from 0 to 50 for the years 2000 to 2050 and from 51 to 99 for the years 1951 to 1999. This is according to the British Standards Institution (BSi) DISC PD2000-1:1998 "A Definition of Year 2000 Conformance Requirements" Rule 3 (b).

Figure 8 – Coding of the data type BinaryDate

4.2.1.9 Encoding of a Time Of Day with and without date indication value

- The encoding of a Time Of Day with and without date indication value shall be primitive.
- The Length field shall indicate as a binary number the number of octets in the Time Of Day with and without date indication value.
- The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 9:

bits octets	7	6	5	4	3	2	1	0	
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	Number of ms since midnight
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	Number of days since 01.01.84 only with date indication
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
	msb							lsb	

Figure 9 – Encoding of Time Of Day value

4.2.1.10 Encoding of a Time Difference with and without date indication value

- The encoding of a Time Difference with and without date indication value shall be primitive.
- The Length field shall indicate as a binary number the number of octets in the Time Difference with and without date indication value.
- The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 10:

bits octets	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}	ms
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	days
6	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	only with date indication
	msb							lsb	

Figure 10 – Encoding of Time Difference value

4.2.1.11 Encoding of a Network Time value

- The encoding of a Network Time value shall be primitive.
- The Length field shall indicate as a binary number the number of octets in the Time Difference value.
- The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 11.

bits octets	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}	Seconds since 1.1.1900 0.00,00 or since 7.2. 2036 6.28,16 when time value less than 0x9dff4400.00000000
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
–	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Fraction Part of seconds one unit is $1/(2^{32})$ s
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}	
	msb							lsb	

Figure 11 – Encoding of Network Time value

4.2.1.12 Encoding of a Network Time Difference value

- The encoding of a Network Time Difference value shall be primitive.
- The Length field shall indicate as a binary number the number of octets in the Time Difference value.
- The ContentsOctets shall be equal in value to the octets in the data value, as shown in Figure 12.

bits octets	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}	Seconds as Integer32
2	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}	
3	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	
4	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	
5	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	Fraction Part of seconds one unit is $1/(2^{32})$ s
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}	
	msb							lsb	

Figure 12 – Encoding of Network Time Difference value

4.2.1.13 Encoding of a NULL value

- The encoding of a NULL value shall be primitive.
- The ContentsOctet shall be omitted.

4.2.1.14 Encoding of a NIL value

- The encoding of a NIL value shall be primitive.
- Each ContentsOctet shall be set to 0x00.

4.2.1.15 Encoding of an UUID

- The encoding of this data type shall be according to Publication C706 of The Open Group.

NOTE The content of Publication C706 is DCE RPC V1.1.

4.2.2 Coding section related to common basic fields

4.2.2.1 Overview

The common fields of 4.2.2 are part of the RTC-PDU and RTA-PDU.

4.2.2.2 Coding of the DLPDU field SourceAddress

This field shall be coded as data type OctetString[6]. The value of the field SourceAddress shall be according to IEEE 802 MAC address, to IEEE 802-1D-2004, Clause 7, and to Table 7.

Table 7 – SourceAddress

PDU	Meaning
RTC-PDU, RTA-PDU, UDP-RTC-PDU, UDP-RTA-PDU, CL-RPC-PDU, ICMP-PDU, DCP-PDU	The interface MAC address is used
MRP-PDU, MRRT-PDU, LLDP-PDU, PTCP-PDU	The port MAC address is used

4.2.2.3 Coding of the DLPDU field DestinationAddress

This field shall be coded as data type OctetString[6]. The value of the field DestinationAddress shall be according to IEEE 802 MAC address.

For DCP-PDUs, this field shall be coded according to Table 8. For all other DCP-PDUs multicast or broadcast addresses shall not be used.

Table 8 – DCP_MulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-00	With FrameID=0xFEFE used for DCP-Identify-ReqPDU
01-0E-CF	00-00-01	With FrameID=0xFEFC used for DCP-Hello-ReqPDU
01-0E-CF	00-00-02 – 00-00-FF	Reserved for other applications

For PTCP-PDUs, the value shall be set according to the Table 9.

Table 9 – PTCP_MulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-01-00 – 00-01-01	Reserved for other applications
01-0E-CF	00-01-02	In conjunction with PTCP-RTCSyncPDU and FrameID(=0x0080) used for clock synchronization
01-0E-CF	00-01-03 – 00-03-FF	Reserved for other applications
01-0E-CF	00-04-00	In conjunction with PTCP-RTASyncPDU, PTCP-AnnouncePDU and FrameID(=0x0000, =0xFF00) used for clock synchronization
01-0E-CF	00-04-01	In conjunction with PTCP-RTASyncPDU, PTCP-AnnouncePDU and FrameID(=0x0001, =0xFF01) used for time synchronization
01-0E-CF	00-04-xx	In conjunction with PTCP-RTASyncPDU, PTCP-AnnouncePDU and FrameID(=0x00xx, =0xFFxx) used for synchronization
01-0E-CF	00-04-1F	In conjunction with PTCP-RTASyncPDU, PTCP-AnnouncePDU and FrameID(=0x001F, =0xFF1F) used for synchronization
01-0E-CF	00-04-20	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF20) used for clock synchronization
01-0E-CF	00-04-21	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF21) used for time synchronization
01-0E-CF	00-04-xx	In conjunction PTCP-FollowUpPDU and FrameID(=0xFFxx) used for synchronization
01-0E-CF	00-04-3F	In conjunction with PTCP-FollowUpPDU and FrameID(=0xFF3F) used for synchronization
01-0E-CF	00-04-40 – FF-FF-FF	Reserved for other applications
01-80-C2	00-00-0E	In conjunction with PTCP-DelayReqPDU and FrameID(=0xFF40), PTCP-DelayResPDU with follow up and FrameID(=0xFF41), PTCP-DelayFuResPDU and FrameID(=0xFF42) and PTCP-DelayResPDU whithout follow up and FrameID (=0xFF43) used for delay messurement

NOTE Octet 1 contains the Individual/Group Address Bit (LSB).

The IEEE Organizationally Unique Identifier for MRP is 00-15-4E. It shall be set according to the Table 10.

Table 10 – MRP OUI

Value for OUI (hexadecimal)	Meaning
00-15-4E	Global administered individual unicast
01-15-4E	Global administered group (multicast) address
02-15-4E	Local administered individual unicast
03-15-4E	Local administered group (multicast) address

For MRP-PDUs, the value shall be set according to the Table 11 and for bumpless extension according to Table 12.

Table 11 – MRPMulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-15-4E	00-00-00	Reserved
01-15-4E	00-00-01	MC_Test, used for media redundancy test frames
01-15-4E	00-00-02	MC_CONTROL, used for media redundancy link change and topology change frames
01-15-4E	00-00-03 – FF-FF-FF	Reserved

NOTE Octet 1 contains the Individual/Group Address Bit (LSB).

Table 12 – MRRTMulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-05-00	MRP extension for bumpless media redundancy (MRRT-PDU)

NOTE Octet 1 contains the Individual/Group Address Bit (LSB).

4.2.2.4 Coding of the field LT

This field shall be coded as data type Unsigned16 with the values according to IEEE 802.3. This specification uses the values according to Table 13.

Table 13 – LT (Length/Type)

Value (hexadecimal)	Meaning
0x0800	IP (UDP, RPC, SNMP, ICMP)
0x0806	ARP
0x8100	Tag Control Information
0x8892	RTC, RTA, DCP, PTCP, MRRT
0x88E3	MRP
0x88CC	LLDP

4.2.2.5 Coding of the field TagControlInformation

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 11: TagControlInformation.VLAN_Id
This field shall be coded according to IEEE 802.1Q.

NOTE It is recommended to use the VLAN_Id 0 that means no use of VLANs.

Bit 12: TagControlInformation.CanonicalFormatIdentificator
This field shall be coded according to IEEE 802.1Q.

NOTE CFI is constant 0.

Bit 13 – 15: TagControlInformation.Priority
This field shall be coded according to IEEE 802.1Q.

This field shall be coded with the values according to Table 14.

Table 14 – TagControlInformation.Priority

Value (hexadecimal)	Meaning
0x00	IP, DCP
0x01	Reserved
0x02	Reserved
0x03	Reserved
0x04	Reserved
0x05	Low prior RTA_CLASS_1or RTA_CLASS_UDP
0x06	RT_CLASS_UDP, RT_CLASS_1, RT_CLASS_2 high prior RTA_CLASS_1 or RTA_CLASS_UDP
0x07	MRP MRRT

NOTE If the used IP implementation does not provide Tag Control Information then the RTA_CLASS_UDP and the RT_CLASS_UDP is send without Tag Control Information.

4.2.2.6 Coding of the field FrameID

This field shall be coded as data type Unsigned16 with the values according to Table 15, Table 16, Table 17, Table 18, Table 19, Table 20, Table 21, Table 22 and Table 23. This field identifies the structure and the type of the APDU.

Table 15 – FrameID range 1

Value (hexadecimal)	Meaning	Use
0x0000	PTCP-RTASyncPDU	Precision transparent clock protocol synchronization without follow up Send Clock and Phase Synchronization
0x0001	PTCP-RTASyncPDU	Precision transparent clock protocol synchronization without follow up Time Synchronization
0x0003 – 0x001F	Reserved	Reserved
0x0020	PTCP-RTASyncPDU	Precision transparent clock protocol synchronization with follow up Send Clock and Phase Synchronization
0x0021	PTCP-RTASyncPDU	Precision transparent clock protocol synchronization with follow up Time Synchronization
0x0022 – 0x007F	Reserved	Reserved

Table 16 – FrameID range 2

Value (hexadecimal)	Meaning	Use
0x0080	PTCP-RTCSyncPDU for RT_CLASS_3	RT_CLASS_3 syncronisation
0x0081 – 0x00FF	Reserved	Reserved

Table 17 – FrameID range 3

Value (hexadecimal)	Meaning	Use
0x0100 – 0x7FFF	Dedicated to RT_CLASS_3 unicast and multicast	RT_CLASS_3

Table 18 – FrameID range 4

Value (hexadecimal)	Meaning	Use
0x8000 – 0xBBFF	Dedicated to RT_CLASS_2 unicast	RT_CLASS_2
0xBC00 – 0xBFFF	Dedicated to RT_CLASS_2 multicast	RT_CLASS_2

Table 19 – FrameID range 5

Value (hexadecimal)	Meaning	Use
0xC000 – 0xF7FF	Dedicated to RT_CLASS_UDP unicast	RT_CLASS_UDP (recommended) and RT_CLASS_1 (legacy); disjunct Frame IDs shall be used if concurrent usage in different ARs
0xF800 – 0xFBFF	Dedicated to RT_CLASS_UDP multicast	RT_CLASS_UDP (recommended) and RT_CLASS_1 (legacy); disjunct Frame IDs shall be used if concurrent usage in different ARs

Table 20 – FrameID range 6

Value (hexadecimal)	Meaning	Use
0xFC00	Reserved	
0xFC01	Alarm High	RTA_CLASS_1 and RTA_CLASS_UDP
0xFC02 – 0xFDFF	Reserved	
0xFE00	Reserved	
0xFE01	Alarm Low	RTA_CLASS_1 and RTA_CLASS_UDP
0xFE02 – 0xFEFB	Reserved	
0xFEFC	DCP-Hello-ReqPDU	DCP
0xFEFD	DCP-Get-ReqPDU, DCP-Get-ResPDU, DCP-Set-ReqPDU, DCP-Set-ResPDU	DCP
0xFEFE	DCP-Identify-ReqPDU	DCP
0xFEFF	DCP-Identify-ResPDU	DCP

Table 21 – FrameID range 7

Value (hexadecimal)	Meaning	Use
0xFF00	PTCP-AnnouncePDU (clock)	Precision transparent clock announce protocol Isochronous application, send clock, and phase synchronization
0xFF01	PTCP-AnnouncePDU (time)	Precision transparent clock announce protocol Time synchronization
0xFF02 – 0xFF1F	Reserved	
0xFF20	PTCP-FollowUpPDU (clock)	Send Clock and phase synchronization (PTCP-FollowUpPDU)
0xFF21	PTCP-FollowUpPDU (time)	Time synchronization (PTCP-FollowUpPDU)
0xFF22 – 0xFF3F	Reserved	
0xFF40	PTCP-DelayReqPDU	For delay messurement
0xFF41	PTCP-DelayResPDU	For delay measurement with follow up
0xFF42	PTCP-DelayFuResPDU	For delay measurement with follow up
0xFF43	PTCP-DelayResPDU	For delay measurement without follow up
0xFF44 – 0xFF5F	Reserved	

Table 22 – FrameID range 8

Value (hexadecimal)	Meaning	Use
0xFF60	MRRT	Bumpless media redundancy realtime activation protocol
0xFF61 – 0xFF6F	Reserved	

Table 23 – FrameID range 9

Value (hexadecimal)	Meaning	Use
0xFF70 – 0xFFFF	Reserved	

4.3 Discovery and basic configuration

4.3.1 DCP syntax description

4.3.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.3.1.2 DCP APDU abstract syntax

Table 24 defines the abstract syntax of the DCP PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 24 – DCP APDU syntax

APDU name	APDU structure
DCP-PDU	DCP-Get-ReqPDU ^ DCP-Set-ReqPDU ^ DCP-Get-ResPDU ^ DCP-Set-ResPDU ^ DCP-Identify-ReqPDU ^ DCP-Identify-ResPDU ^ DCP-Hello-ReqPDU
DCP-Identify-ReqPDU	DCP-IdentifyFilter-ReqPDU ^ DCP-IdentifyAll-ReqPDU
DCP-IdentifyFilter-ReqPDU	DCP-MC-Header, [NameOfStationBlock] ^ [AliasNameBlock], IdentifyReqBlock* [a]
DCP-IdentifyAll-ReqPDU	DCP-MC-Header, AllSelectorBlock
DCP-Hello-ReqPDU	DCP-UC-Header, NameOfStationBlockRes, { IPParameterBlockRes, DeviceIDBlockRes, DeviceOptionsBlockRes, DeviceRoleBlockRes, DeviceInitiativeBlockRes }
DCP-Identify-ResPDU	DCP-UC-Header, { IdentifyResBlock * [b], NameOfStationBlockRes [c], IPParameterBlockRes [c], DeviceIDBlockRes [c], DeviceOptionsBlockRes [c], DeviceRoleBlockRes [c], [DeviceInitiativeBlockRes] [d] }
DCP-Get-ReqPDU	DCP-UC-Header, GetReqBlock*
DCP-Get-ResPDU	DCP-UC-Header, (GetResBlock ^ GetNegResBlock)*
DCP-Set-ReqPDU	DCP-UC-Header, [StartTransactionBlock, DCPBlocklength, BlockQualifier], FactoryResetBlock ^ SetReqBlock*, [StopTransactionBlock, DCPBlocklength, BlockQualifier]
DCP-Set-ResPDU	DCP-UC-Header, (SetResBlock ^ SetNegResBlock)*

[a] The content of the field value of the IdentifyReqBlock of the PDU shall be interpreted as a filter at the receiving instance. All included DataBlocks shall be operated with a logical AND and it shall only responded with a DCP-Identify-ResPDU if all filter criteria match.

[b] The content of the field value of the IdentifyReqBlock of the PDU shall be of the same option and suboption of the requested once.

[c] The field shall only be present if it is not already part of the IdentifyResBlock.

[d] The field shall only be present if the usage of DCP-Hello-ReqPDU is activated.

Table 25 defines structures for substitutions of elements of the APDU structures shown in Table 24.

Table 25 – DCP substitutions

Substitution name	Structure
DCP-MC-Header	ServiceID, ServiceType, Xid, ResponseDelayFactor, DCPDataLength
DCP-UC-Header	ServiceID, ServiceType, Xid, Padding* [a], DCPDataLength [a] Number of Padding octets shall be 2.
IdentifyReqBlock	DeviceRoleBlock ^ DeviceVendorBlock ^ DeviceIDBlock ^ DeviceOptionsBlock ^ MACAddressBlock ^ IPParameterBlock ^ DHCPParameterBlock ^ ManufacturerSpecificParameterBlock
IdentifyResBlock	NameOfStationBlockRes ^ DeviceRoleBlockRes ^ DeviceVendorBlockRes ^ DeviceIDBlockRes ^ DeviceOptionsBlockRes ^ MACAddressBlockRes ^ IPParameterBlockRes ^ DHCPParameterBlockRes ^ ManufacturerSpecificParameterBlockRes ^ AliasNameBlockRes
GetReqBlock	NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ MACAddressType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType ^ AllSelectorType
GetResBlock	NameOfStationBlockRes ^ DeviceRoleBlockRes ^ DeviceVendorBlockRes ^ DeviceIDBlockRes ^ DeviceOptionsBlockRes ^ MACAddressBlockRes ^ IPParameterBlockRes ^ DHCPParameterBlockRes ^ ManufacturerSpecificParameterBlockRes
GetNegResBlock	OptionControl, SuboptionResponse, DCPBlocklength, NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ MACAddressType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType, BlockError, [AddDataValue]
SetReqBlock	SignalType ^ NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType, DCPBlocklength, BlockQualifier, SignalValue ^ NameOfStationValue ^ DeviceRoleValue ^ DeviceVendorValue ^ DeviceIDValue ^ DeviceOptionsValue ^ IPParameterValue ^ DHCPParameterValue ^ ManufacturerSpecificParameterValue
SetResBlock	OptionControl, SuboptionResponse, DCPBlocklength, NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType ^ SignalType ^ FactoryResetType ^ StartTransactionType ^ StopTransactionType, BlockError(=0), [AddDataValue]
SetNegResBlock	OptionControl, SuboptionResponse, DCPBlocklength, NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType ^ SignalType ^ FactoryResetType ^ StartTransactionType ^ StopTransactionType, BlockError(<>0), [AddDataValue]
DeviceInitiativeType	DeviceInitiativeOption, DeviceInitiativeSuboption
DeviceInitiativeBlock	DeviceInitiativeType, DCPBlocklength, DeviceInitiativeValue
DeviceInitiativeBlockRes	DeviceInitiativeType, DCPBlocklength, BlockInfo, DeviceInitiativeValue
StartTransactionType	OptionControl, SuboptionStart
StartTransactionBlock	StartTransactionType, DCPBlocklength
StopTransactionType	OptionControl, SuboptionStop
StopTransactionBlock	StopTransactionType, DCPBlocklength
SignalType	OptionControl, SuboptionSignal
SignalBlock	SignalType, DCPBlocklength(=2), SignalValue
FactoryResetType	OptionControl, SuboptionFactoryReset
FactoryResetBlock	FactoryResetType, DCPBlocklength(=0)

Substitution name	Structure
NameOfStationType	DevicePropertiesOption, SuboptionNameOfStation
NameOfStationBlock	NameOfStationType, DCPBlocklength, NameOfStationValue
NameOfStationBlockRes	NameOfStationType, DCPBlocklength, BlockInfo, NameOfStationValue
AliasNameType	DevicePropertiesOption, SuboptionAliasName
AliasNameBlock	AliasNameType, DCPBlocklength, AliasNameValue
AliasNameBlockRes	AliasNameType, DCPBlocklength, BlockInfo, AliasNameValue
DeviceRoleType	DevicePropertiesOption, SuboptionDeviceRole
DeviceRoleBlock	DeviceRoleType, DCPBlocklength, DeviceRoleValue
DeviceRoleBlockRes	DeviceRoleType, DCPBlocklength, BlockInfo, DeviceRoleValue
DeviceVendorType	DevicePropertiesOption, SuboptionDeviceVendor
DeviceVendorBlock	DeviceVendorType, DCPBlocklength, DeviceVendorValue
DeviceVendorBlockRes	DeviceVendorType, DCPBlocklength, BlockInfo, DeviceVendorValue
DeviceIDType	DevicePropertiesOption, SuboptionDeviceID
DeviceIDBlock	DeviceIDType, DCPBlocklength, DeviceIDValue
DeviceIDBlockRes	DeviceIDType, DCPBlocklength, BlockInfo, DeviceIDValue
DeviceIDValue	VendorIDHigh, VendorIDLow, DeviceIDHigh, DeviceIDLow
DeviceOptionsType	DevicePropertiesOption, SuboptionDeviceOptions
DeviceOptionsBlock	DeviceOptionsType, DCPBlocklength, DeviceOptionsValue
DeviceOptionsBlockRes	DeviceOptionsType, DCPBlocklength, BlockInfo, DeviceOptionsValue
DeviceOptionsValue	[NameOfStationType ^ DeviceRoleType ^ DeviceVendorType ^ DeviceIDType ^ DeviceOptionsType ^ MACAddressType ^ IPParameterType ^ DHCPParameterType ^ ManufacturerSpecificParameterType]*
MACAddressType	IPOption, SuboptionMACAddress
MACAddressBlock	MACAddressType, DCPBlocklength, MACAddressValue
MACAddressBlockRes	MACAddressType, DCPBlocklength, BlockInfo, MACAddressValue
IPParameterType	IPOption, SuboptionIPParameter
IPParameterBlock	IPParameterType, DCPBlocklength, IPParameterValue
IPParameterBlockRes	IPParameterType, DCPBlocklength, BlockInfo, IPParameterValue
DHCPParameterType	DHCPOption, SuboptionDHCP
DHCPParameterBlock	DHCPParameterType, DCPBlocklength, DHCPParameterValue
DHCPParameterBlockRes	DHCPParameterType, DCPBlocklength, BlockInfo, DHCPParameterValue
ManufacturerSpecificParameterType	ManufacturerSpecificOption, SuboptionManufacturerSpecific
ManufacturerSpecificParameterBlock	ManufacturerSpecificParameterType, DCPBlocklength, ManufacturerSpecificParameterValue
ManufacturerSpecificParameterBlockRes	ManufacturerSpecificParameterType, DCPBlocklength, BlockInfo, ManufacturerSpecificParameterValue
AllSelectorType	AllSelectorOption, SuboptionAllSelector
AllSelectorBlock	AllSelectorType, DCPBlocklength
DeviceRoleValue	DeviceRoleDetails, Padding
IPParameterValue	IPAddress, Subnetmask, StandardGateway
DHCPParameterValue	SuboptionDHCP, DHCPParameterLength, DHCPParameterData
ManufacturerSpecificParameterValue	ManufacturerOUI, ManufacturerSpecificString

4.3.1.3 Coding section related to header fields

4.3.1.3.1 Coding of the field ServiceID

This field shall be coded as data type Unsigned8 and shall contain the values as described in Table 26. They shall be set in the appropriate PDU.

Table 26 – ServiceID

Value (hexadecimal)	Meaning
0x00 – 0x02	Reserved
0x03	Get
0x04	Set
0x05	Identify
0x06	Hello
0x07 – 0xFF	Reserved

4.3.1.3.2 Coding of the field ServiceType

This field shall be coded as data type Unsigned8 and shall contain the values as described in Table 27 and Table 28. They shall be set in the appropriate PDU.

Table 27 – ServiceType for request

Bit	Usage	Value (binary)	Meaning
Bit 0	Selection	0	Request
Bit 1 – 7	Reserved		Shall be set to zero

Table 28 – ServiceType for response

Bit	Usage	Value (binary)	Meaning
Bit 0	Selection	1	Response
Bit 1	Reserved		Shall be set to zero
Bit 2	Response	0	Success
		1	Request not supported
Bit 3 – 7	Reserved		Shall be set to zero

4.3.1.3.3 Coding of the field Xid

This field shall be coded as data type Unsigned32. It shall contain a transaction ID chosen by the client to associate requests and responses between a client and a server.

4.3.1.3.4 Coding of the field DCPDataLength

This field shall be coded as data type Unsigned16. It shall contain the total length of data followed the DCP-UC-Header or DCP-MC-Header in octets. The maximum length of DCP Data is 1 432 bytes.

4.3.1.3.5 Coding of the field ResponseDelayFactor

This field shall be coded as data type Unsigned16. It shall contain a delay factor that shall be used by the server to calculate its individual response delay time. The minimum response delay time is between 10 ms and 64 s. The server shall calculate the response delay time on a 10 ms base as described below.

The last two bytes of the IEEE 802 MAC address shall be used as random number K. Octet 6 of the MAC address represents the low-order byte and octet 5 the high-order byte.

The response delay time is calculated according to the following equations.

$$\text{Spread} = \text{K mod ResponseDelayFactor} \tag{1}$$

Spread not equal zero:

$$\text{Minimal Response Delay} = 10\ \text{ms} \times \text{Spread} \tag{2}$$

Spread equal zero:

$$\text{Minimal Response Delay} = 0\ \text{ms} \tag{3}$$

Allowed values: 1 – 6 400

4.3.1.4 Coding section of block fields

4.3.1.4.1 General

The block fields are parted into option, suboption, block length, block info, and value. Table 29 shows the list of available options and Table 30 shows the list of available suboptions.

Table 29 – List of options

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	IPOption
0x02	DevicePropertiesOption
0x03	DHCPOption
0x04	Reserved
0x05	OptionControl
0x06	DeviceInitiativeOption
0x07 – 0x7F	Reserved
0x80 – 0xFE	ManufacturerSpecificOption
0xFF	AllSelectorOption

Table 30 – List of suboptions

Value (hexadecimal)	Meaning
0x01	SuboptionStart
0x02	SuboptionStop
0x03	SuboptionSignal
0x04	SuboptionResponse
0x05	SuboptionFactoryReset
0x01	DeviceInitiativeSuboption
0x01	SuboptionMACAddress
0x02	SuboptionIPParameter
0x01	SuboptionDeviceVendor
0x02	SuboptionNameOfStation
0x03	SuboptionDeviceID
0x04	SuboptionDeviceRole
0x05	SuboptionDeviceOptions
0x06	SuboptionAliasName
0xFF	SuboptionAllSelector
0x00 – 0xFF	SuboptionManufacturerSpecific
See Table 31	SuboptionDHCP

4.3.1.4.2 Coding of the field IPOption

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.3 Coding of the field DevicePropertiesOption

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.4 Coding of the field DHCPOption

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.5 Coding of the field OptionControl

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.6 Coding of the field DeviceInitiativeOption

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.7 Coding of the field ManufacturerSpecificOption

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.8 Coding of the field AllSelectorOption

This field shall be coded as data type Unsigned8 according to Table 29.

4.3.1.4.9 Coding of the field SuboptionStart

This field shall be coded as data type Unsigned8 according to Table 30.

The suboption StartTransaction shall be coded with DCPBlocklength zero. There are no additional data for this suboption.

4.3.1.4.10 Coding of the field SuboptionStop

This field shall be coded as data type Unsigned8 according to Table 30.

The suboption StopTransaction shall be coded with DCPBlocklength zero. There are no additional data for this suboption.

4.3.1.4.11 Coding of the field SuboptionSignal

This field shall be coded as data type Unsigned8 according to Table 30.

The suboption Signal shall be coded with DCPBlocklength zero. There are no additional data for this suboption.

4.3.1.4.12 Coding of the field SuboptionResponse

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.13 Coding of the field SuboptionFactoryReset

This field shall be coded as data type Unsigned8 according to Table 30.

The suboption FactoryReset shall be coded with DCPBlocklength zero. There are no additional data for this suboption.

Factory Reset shall permanent set

- The NameOfStation to “”
- The IP address, the subnet mask and the standard gateway to 0.0.0.0
- All other parameters to the manufacturers default value.

4.3.1.4.14 Coding of the field DeviceInitiativeSuboption

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.15 Coding of the field SuboptionMACAddress

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.16 Coding of the field SuboptionIPParameter

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.17 Coding of the field SuboptionDeviceVendor

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.18 Coding of the field SuboptionNameOfStation

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.19 Coding of the field SuboptionDeviceID

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.20 Coding of the field SuboptionDeviceRole

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.21 Coding of the field SuboptionDeviceOptions

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.22 Coding of the field SuboptionAliasName

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.23 Coding of the field SuboptionAllSelector

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.24 Coding of the field SuboptionDHCP

This field shall be coded as data type Unsigned8. The allowed values shall be according Table 31.

Table 31 – SuboptionDHCP

Value (decimal)	Data Length	Description	References
0 – 11		Reserved [a]	
12	1 +	Host Name [d]	RFC 2132
13 – 42		Reserved [a]	
43	1 +	Vendor specific information	RFC 2132
44 – 53		Reserved [a]	
54	4	Server identifier	RFC 2132
55	1 +	Parameter request list [c]	RFC 2132
56 – 59		Reserved [a]	
60	1 +	Class-identifier	RFC 2132
61	2 +	DHCP client identifier	RFC 2132
62 – 80		Reserved [a]	
81	1 +	FQDN, Fully Qualified Domain Name [d]	
82 – 96		Reserved [a]	
97	Variable	UUID/GUID-based Client (=addressed Station) Identifier	
98 – 254		Reserved [a]	
255	1	Control DHCP for address resolution 0: Don't use DHCP 1: Don't use DHCP and reset all DHCP options 2: Use DHCP [b] for address resolution	

[a] The reserved options should be used in the same meaning as they are defined in the corresponding RFCs for DHCP.

[b] This option should be the last in a list of DHCP options in a set request.

[c] In this option, at least the parameters 1 (subnet mask) and 3 (router) should be requested.

[d] The use of these suboptions is still not fixed and subject to change in later revisions.

4.3.1.4.25 Coding of the field SuboptionManufacturerSpecific

This field shall be coded as data type Unsigned8 according to Table 30.

4.3.1.4.26 Coding of the field DCPBlocklength

This field shall be coded as data type Unsigned16 and shall contain the length of the specific data of the suboption without counting padding octets.

4.3.1.4.27 Coding of the field BlockQualifier

This field shall be coded as data type Unsigned16.

Allowed values with the option IP suboption IP parameter shall be according to Table 32.

Table 32 – BlockQualifier with option IP

Bit	Value (binary)	Meaning
Bit 0	0	Use IP Address, Subnetmask, and Default Gateway temporary and delete permanent store values Set permanent IP Address, Subnetmask, and Default Gateway to 0.0.0.0
	1	Save IP Address, Subnetmask, and Default Gateway permanent and use it after restart
Bit 1 – 15		Reserved

NOTE The IP suboption MAC Address is not topic of change via DCP and therefore no DataQualifier is defined here.

Allowed values with the option DeviceProperties, DHCP, and ManufacturerSpecific shall be according to Table 33.

Table 33 – BlockQualifier with option DeviceProperties, DHCP, and ManufacturerSpecific

Bit	Value (binary)	Meaning
Bit 0	0	Use the value of the option/suboption temporary
	1	Save the value of the option/suboption permanent and use it also after restart
Bit 1 – 15		Reserved

4.3.1.4.28 Coding of the field BlockError

This field shall be coded as data type Unsigned8 and shall be set according to Table 34.

Table 34 – BlockError

Value (hexadecimal)	Meaning	Usage
0x00	No error	Positive response Parameter delivered and accepted
0x01	Option not supported	If optional option is not supported
0x02	Suboption not supported or no DataSet available	If optional suboption is not supported
0x03	Suboption not set	If suboption could not be set for local reasons
0x04	Resource error	If there is a temporary resource error within the server
0x05	SET not possible by local reasons	If Set service is not possible for local reasons
0x06	In operation, SET not possible	If Set service is not possible because of application operation
0x07 – 0xFF	Reserved	Shall not be used

4.3.1.4.29 Coding of the field BlockInfo

This field shall be coded as data type Unsigned16. The allowed values shall be set according to Table 35, Table 36, Table 37, and Table 38.

Table 35 – BlockInfo for SuboptionIPParameter

Bit	Meaning
Bit 0 – 1	See Table 36
Bit 2 – 6	Reserved
Bit 7	See Table 37
Bit 8 – 15	Reserved

Table 36 – Bit 1 and Bit 0 of BlockInfo for SuboptionIPParameter

Bit 1	Bit 0	Meaning for SuboptionIPParameter
0	0	No IP parameter
0	1	IP Parameter set
1	0	IP parameter set via DHCP
1	1	Reserved

Table 37 – Bit 7 of BlockInfo for SuboptionIPParameter

Bit 7	Meaning for SuboptionIPParameter
0	No IP address conflict detected
1	IP address currently not active because IP address conflict detected

Table 38 – BlockInfo for all other suboptions

Bit	Meaning
Bit 0 – 15	Reserved

4.3.1.4.30 Coding of the field DeviceInitiativeValue

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: DeviceInitiativeValue.Hello
This field shall be set according to the Table 39.

Table 39 – DeviceInitiativeValue

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Device does not issue a DCP-Hello-ReqPDU after power on.
0x01	ON	Device does issue a DCP-Hello-ReqPDU after power on.

Bit 1 – 15: DeviceInitiativeValue.reserved
This field shall be set according to 3.7.3.2.

4.3.1.4.31 Coding of the field SignalValue

This field shall be coded as data type Unsigned16 according to Table 40.

Table 40 – SignalValue

Value (hexadecimal)	Meaning	Usage
0x0100	Flash Once	Flash the Ethernet LINK LED or an alternative signalization with duration of 3 s with a frequency of 2 Hz (500 ms on, 500 ms off).

4.3.1.4.32 Coding of the field NameOfStationValue

4.3.1.4.32.1 Encoding

This field shall be coded as data type OctetString with 1 to 240 octets. The following syntax applies:

- 1 or more labels, separated by [.]
- Total length is 1 to 240
- Label length is 1 to 63
- Labels consist of [a-z0-9-]
- Labels do not start with [-]
- Labels do not end with [-]
- The first label does not start with "port-xyz" or "port-xyz-abcde" with a,b,c,d,e, x, y, z = 0...9
- Station-names do not have the form n.n.n.n, n = 0...999
- Labels do only start with 'xn-' if RFC 3490 is applied

Furthermore, the definition of RFC 3490 shall be applied.

EXAMPLE 1 “device-1.machine-1.plant-1.vendor“

EXAMPLE 2 “device-1.bögeholz“ is coded as “device-1.xn-bgeholz-90a“

NOTE The field NameOfStationValue is not terminated by zero.

Each NameOfStation shall be unique in case a device is equipped with more than one interface module.

4.3.1.4.32.2 Semantics with Identify service

The following rules shall be applied:

- Name of station with length 1 to 240 octects:
  Only the device shall answer which has exactly got this name. In the comparison of the two names on equality the DNS conventions shall be considered.
- Name of station with length == 0:
  Only those devices shall answer, which have not received a station name yet.

The network representation of the NameOfStation shall be according to 4.3.1.4.32.1.

4.3.1.4.32.3 Semantics with Set service

The following rules shall be applied:

- Name of station with length 1 to 240 octects:
  The station shall set its name according to the NameOfStationValue.
- Name of station with length == 0:
  The station shall delete its stored name.

The network representation of the NameOfStation shall be according to 4.3.1.4.32.1.

4.3.1.4.33 Coding of the field AliasNameValue

4.3.1.4.33.1 Encoding

The field shall be coded as OctetString. The content shall be the concatenation of the content of the fields LLDP_PortID and LLDP_ChassisID.

AliasNameValue = LLDP_PortID + "." + LLDP_ChassisID (4)

4.3.1.4.33.2 Semantics with Identify service

The following rules shall be applied:

- Alias name with length != 0:
  Only the device shall answer which has exactly got this additional alias name. In the comparison of the two names on equality the DNS conventions shall be considered. The device shall not answer with the AliasName. Instead it shall use the current NameOfStation (even if the NameOfStation is not set).
- Name of the station with length == 0:
  Not allowed in a request.

NOTE The AliasName is an unique alternative name for a device. The alias name is derived from topology or neighbourhood information. The alias name of station cannot be set or get through means of DCP.

4.3.1.4.34 Coding of the field DeviceRoleDetails

This field shall be coded as data type Unsigned8 with the values according to Table 41.

Table 41 – DeviceRoleDetails

Bit	Meaning	Usage
Bit 0	PNIO Device	Device has the specific role
Bit 1	PNIO Controller	Device has the specific role
Bit 2	PNIO Multidevice	Device has the specific role
Bit 3	PNIO Supervisor	Device has the specific role
Bit 4 – 7	Reserved	

4.3.1.4.35 Coding of the field MACAddressValue

This field shall be coded as data type OctetString[6]. The value of the field shall be according to IEEE 802 MAC address.

The least significant bit of the first Octet shall be zero. The address assignment rules of IANA shall be applied.

4.3.1.4.36 Coding of the field IPAddress

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to RFC 791 and to Table 42.

The value zero in all octets shall be use to delete a currently set IP address.

Table 42 – IPAddress

Value (hexadecimal)	Meaning
0.0.0.0	No IP address assigned
Other	IP address assigned

4.3.1.4.37 Coding of the field Subnetmask

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to RFC 791 and to Table 43.

Table 43 – Subnetmask

Value (hexadecimal)	Meaning
0.0.0.0	No subnet mask assigned
Other	Subnet mask assigned

4.3.1.4.38 Coding of the field StandardGateway

This field shall be coded as data type OctetString[4]. The encoding of the fields shall be according to RFC 791 and to Table 44.

If there is no configuration of the Default-router the value shall be set to the same value as the field IPAddress. In this case, the application shall set up the local IP implementation to avoid IP routing.

Table 44 – StandardGateway

Value (hexadecimal)	Meaning
0.0.0.0	No standard gateway assigned
Current IP address	No standard gateway assigned
Other	Standard gateway assigned

If no default router is configured, the value of StandardGateway in the GetResBlock shall be set to the same value as the field IPAddress.

4.3.1.4.39 Coding of the fields DHCPParameterValue using DHCP Option 61

The client identifier can be used by a DHCP client to request its IP parameters from a DHCP server. A device shall be able to use the client identifier from the data set in the following way:

4.3.1.4.39.1 Coding of the field DHCPParameterLength

This field shall be coded as data type Unsigned16 and shall contain the length of the specific subsequent data.

4.3.1.4.39.2 Use of MACAddress as client identifier

If the device shall obtain its IP parameters using the DHCP protocol and use the Mac address as a client identifier, then the option 61 shall have the following values:

SuboptionDHCP = 61

DHCPParameterLength = 1

DHCPParameterData = 1

4.3.1.4.39.3 Use of NameOfStation as client identifier

If the device shall obtain its IP parameters using the DHCP protocol and use the NameofStation as client identifier the option 61 shall have the following values (in addition to RFC 2132):

SuboptionDHCP = 61

DHCPParameterLength = 1

DHCPParameterData = 0

If the name of station of the device has been changed the DHCP request shall automatically use the new NameOfStation in the next DHCP request.

4.3.1.4.39.4 Use of arbitrary client identifier

If the device shall obtain its IP parameters using the DHCP protocol and use an arbitrary string as client identifier the option 61 shall have the following values:

SuboptionDHCP = 61

DHCPParameterLength => 1

DHCPParameterData[first octet] = 0

DHCPParameterData[second octet …n] =”NameOfStation”

4.3.1.4.40 Coding of the field ManufacturerOUI

This field shall be coded as OctetString[3] with the Organizationally unique identifier (OUI) as defined by IEEE 802.

Note The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <http://standard.ieee.org/regauth>

4.3.1.4.41 Coding of the field ManufacturerSpecificString

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be word aligned. The padding octet shall have the value 0.

4.3.1.4.42 Coding of the field DeviceVendorValue

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be word aligned. The padding octet shall have the value 0.

NOTE This string may contain a model name or an ordering number.

4.3.1.4.43 Coding of the field DHCPParameterData

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be word aligned. The padding octet shall have the value 0.

4.3.1.4.44 Coding of the field AddDataValue

This field shall be coded as data type OctetString and shall contain the values of the related option and suboptions. If the BlockValue has an odd length a padding octet shall be added at the end in order to be word aligned. The padding octet shall have the value 0.

4.3.2 DCP protocol state machines

4.3.2.1 Application relationship monitoring

4.3.2.1.1 Timer

UC Client Timeout
This Timer, restarted on the requestor when sending an unicast request, will terminate waiting for an response. The value shall be taken from the appropriate DCP ASE attribute.

MC Client Timeout
This Timer, restarted when sending a multicast request on the requestor, will terminate waiting for a response. The value shall be taken from the appropriate DCP ASE attribute.

4.3.2.1.2 Intervals of the receiver

Response Delay Time
This monitoring time specifies the time period waiting for DCP-IdentifyResPDUs.

Client Hold Time
This monitoring time specifies the smallest allowed period of time between a response and the release of the client. This is used to allow multiple services executed from a single source without disruption of other nodes. The value shall be taken from the appropriate DCP ASE attribute.

4.3.2.2 Application Relationship Protocol Machines (ARPMs)

4.3.2.2.1 DCPUCS

4.3.2.2.1.1 Primitive definitions

4.3.2.2.1.1.1 Primitives exchanged between DCPUCS and DCP user

The service primitives including their associated parameters issued by DCP user received by DCPUCS and vice versa are described in the DCP ASE in the service definition.

4.3.2.2.1.1.2 Primitives exchanged between DCPUCS and LMPM

The service primitives including their associated parameters issued by DCPUCS received by LMPM and vice versa are described in the LMPM service definition.

4.3.2.2.1.2 State machine description

The state machine gets its initial values from the DCP ASE attributes. Within the OPEN state, DCP Get and Set service requests issued by the DCP user are transformed to Data Link Layer services. After issuing the transmission of data, the WACK state is used to wait for the Response-PDU causing a confirmation primitive to the DCP user.

Local variables of the DCPUCS:

SXID
This local variable contains the Transaction ID XID used to identify services uniquely. The initial value shall be derived from a random number generator.

NOTE The current time value in nanoseconds may be used as a random number.

Pending
This local variable holds the information whether a Set or Get service is pending.

Retry
This local variable contains the retry counter. The maximum value shall be taken from the appropriate DCP ASE attribute.

4.3.2.2.1.3 DCPUCS state table

Table 45 contains the complete description of the DCPUCS state table.

Table 45 – DCPUCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Get.req(CREP, DA, ListOfOptions)** => Pending:= GET DCP-UC-Header.Xid:= SXID A_SDU := DCP-GetReqPDU Store A_SDU Retry:= Max Retry Limit StartTimer(UC Client Timeout) LMPM_A_Data.req (CREP, DA, SA, Prio=0, A_SDU)	WACK
2	OPEN	**DCP_Set.req(CREP, DA, ListOfData, ListOfControlCommands)** => Pending:= SET DCP-UC-Header.Xid:= SXID A_SDU := DCP-SetReqPDU Store A_SDU Retry:= Max Retry Limit StartTimer(UC Client Timeout) LMPM_A_Data.req (CREP, DA, SA, Prio=0, A_SDU)	WACK
3	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** => ignore	OPEN
4	OPEN	**LMPM_A_Data.cnf (CREP, LMPM_status)** => ignore	OPEN
5	OPEN	**Timeout** => ignore	OPEN
6	WACK	**DCP_Get.req(CREP, DA, ListOfOptions)** => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Get.cnf(-) (ERRCLS, ERRCODE)	WACK
7	WACK	**DCP_Set.req(CREP, DA, ListOfData, ListOfControlCommands)** => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Set.cnf(-) (ERRCLS, ERRCODE)	WACK
8	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Get-ResPDU && Pending == GET && DCP-Header.xid == SXID => SXID:= (SXID+1) & 0xffffffff DCP_Get.cnf(+) (ListOfData)	OPEN
9	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Get-ResPDU && (Pending != GET || DCP-Header.xid != SXID) => ignore	WACK

#	Current State	Event /Condition =>Action	Next State
10	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Set-ResPDU && Pending == SET && DCP-Header.xid == SXID => SXID:= (SXID+1) & 0xffffffff DCP_Set.cnf(+) (CREP, ListOfData, ListOfControlCommands)	OPEN
11	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Set-ResPDU && (Pending != SET || DCP-Header.xid != SXID) => ignore	WACK
12	WACK	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status == OK => ignore	WACK
13	WACK	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status != OK && Pending == GET => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer() DCP_Get.cnf(-) (ERRCLS, ERRCODE)	CLOSED
14	WACK	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status != OK && Pending == SET => ERRCLS := PROTOCOL ERRCODE := LMPM StopTimer() DCP_Set.cnf(-) (ERRCLS, ERRCODE)	CLOSED
15	WACK	**Timeout** /Retry != 0 => A_SDU:=from Stored A_SDU Retry:= Retry-1 LMPM_A_Data.req (DA, SA, Prio, A_SDU)	WACK
16	WACK	**Timeout** /Retry == 0 && Pending == GET => ERRCLS := PROTOCOL ERRCODE := TIMEOUT DCP_Get.cnf(-) (ERRCLS, ERRCODE)	OPEN
17	WACK	**Timeout** /Retry == 0 && Pending == SET => ERRCLS := PROTOCOL ERRCODE := TIMEOUT DCP_Set.cnf(-) (ERRCLS, ERRCODE)	OPEN
18	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /!(DCP-Set-ResPDU || DCP-Get-ResPDU) => ignore	WACK

4.3.2.2.2 DCPUCR

4.3.2.2.2.1 Primitive definitions

4.3.2.2.2.1.1 Primitives exchanged between DCPUCR and DCP user

The service primitives including their associated parameters issued by DCP user received by DCPUCR and vice versa are described in the DCP ASE in the service definition.

4.3.2.2.2.1.2 Primitives exchanged between DCPUCR and LMPM

The service primitives including their associated parameters issued by DCPUCS received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.3.2.2.2.2 State machine description

Within the OPEN state, the state machine is waiting for the first LMPM_A_Data service indication. After passing the indication to the user the WACK state is entered. A response service will activate an LMPM_A_Data request and bring the machine back to the OPEN state.

Local variables of the DCPUCR:

SAM
This local variable contains the last valid SA conveyed with a LMPM_A_Data indication with valid data.

SXID
This local variable contains the Transaction ID XID used to identify services uniquely. The value shall be taken from the request PDU.

4.3.2.2.2.3 DCPUCR state table

Table 46 contains the complete description of the DCPUCR state table. A LMPM_A_Data.ind primitive will be accepted if type is DCP-Get-ResPDU or DCP-Set-ResPDU.

Table 46 – DCPUCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Get.rsp(CREP, ListOfData)** => ignore	OPEN
2	OPEN	**DCP_Set.rsp(CREP, ListOfResponse)** => ignore	OPEN
3	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Get-ReqPDU && (SAM == NIL || SAM == SA) => SAM:= SA SXID := DCP_Header.xid StartTimer(Client Hold Time) DCP_Get.ind (SA,ListOfSelectors)	WACK
4	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Set-ReqPDU && (SAM == NIL || SAM == SA) => SAM:= SA SXID := DCP_Header.xid StartTimer(Client Hold Time) DCP_Set.ind (SA,ListOfData)	WACK

#	Current State	Event /Condition =>Action	Next State
5	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Get-ReqPDU && (SAM != NIL && SAM != SA) => ignore	OPEN
6	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Set-ReqPDU && (SAM != NIL && SAM != SA) => ignore	OPEN
7	OPEN	**LMPM_A_Data.cnf (CREP, LMPM_status)** => ignore	OPEN
8	OPEN	**Timeout** => SAM:= NIL	OPEN
9	WACK	**DCP_Get.rsp(CREP, ListOfData)** => DCP_Header.op:= GET DCP_Header.xid:= SXID DA := SAM A_SDU := DCP-Get-ResPDU StopTimer() LMPM_A_Data.req (CREP, DA, SA, Prio, A_SDU)	OPEN
10	WACK	**DCP_Set.rsp(CREP, ListOfResponse)** => DCP_Header.op:= SET DCP_Header.xid:= SXID DA := SAM A_SDU := DCP-Set-ResPDU StopTimer() LMPM_A_Data.req (CREP, DA, SA, Prio, A_SDU)	OPEN
11	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** => ignore	WACK

4.3.2.2.3 DCPMCS

4.3.2.2.3.1 Primitive definitions

4.3.2.2.3.1.1 Primitives exchanged between DCPMCS and DCP user

The service primitives including their associated parameters issued by DCP user received by DCPMCS and vice versa are described in the DCP ASE in the service definition.

4.3.2.2.3.1.2 Primitives exchanged between DCPMCS and LMPM

The service primitives including their associated parameters issued by DCPMCS received by LMPM and vice versa are described in the LMPM service definition.

4.3.2.2.3.2 State machine description

The machine waits in the OPEN state for an Identify service request. After issuing the transmission of the data the WACK state is used to wait for all responses until a certain time.

The timeout causes to issue the Identify contirmation service primitive with all received responses to the DCP user and sets the state machine to the OPEN state.

Local variables of the DCPMCS

SXID
This local variable contains the Transaction ID XID used to Identify services uniquely. The initial value shall be derived from a random number generator.

4.3.2.2.3.3 DCPMCS state table

Table 47 contains the complete description of the DCPMCS state machine. A LMPM_A_Data.ind primitive will be accepted if type is DCP-IdentifyResPDU.

Table 47 – DCPMCS state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Identify.req(ListOfFilter, ResponseDelayFactor)** => DCP_Header.xid:= SXID DA=01:0E:CF:00:00:00 A_SDU := DCP-Identify-ReqPDU StartTimer((round up to next second (ResponseDelayFactor*10 ms)+1 s)) LMPM_A_SDU.req(CREP,DA, SA, VLAN-Prio,A_SDU)	WACK
2	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** => ignore	OPEN
3	OPEN	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status == OK => ignore	OPEN
4	OPEN	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status != OK => ignore	OPEN
5	OPEN	**Timeout** => ignore	OPEN
6	WACK	**DCP_Identify.req(ListOfFilter, ResponseDelayFactor)** => ERRCLS := CTXT ERRCODE := INVALID_STATE DCP_Identify.cnf(-) (CREP,ERRCLS,ERRCODE)	WACK
7	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Identify-ResPDU && DCP-Header.xid == SXID => ListOfDevices.SA=SA ListOfDevices.ListOfData = A_SDU	WACK

#	Current State	Event /Condition =>Action	Next State
8	WACK	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /!(DCP-Identify-ResPDU) || (DCP-Header.xid != SXID) => ignore	WACK
9	WACK	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status == OK => ignore	WACK
10	WACK	**LMPM_A_Data.cnf (CREP, LMPM_status)** /LMPM_status != OK => ERRCLS := PROTOCOL ERRCODE := LMPM DCP_Identify.cnf(-) (ERRCLS,ERRCODE)	OPEN
11	WACK	**Timeout** => SXID:= SXID + 1 mod 0xffffffff ListOfDevices=Stored ListOfDevices DCP_Identify.cnf(+) (ListOfDevices)	OPEN

4.3.2.2.4 DCPMCR

4.3.2.2.4.1 Primitive definitions

4.3.2.2.4.1.1 Primitives exchanged between DCPMCR and DCP user

The service primitives including their associated parameters issued by DCP user received by DCPMCR and vice versa are described in the DCP ASE in the service definition.

4.3.2.2.4.1.2 Primitives exchanged between DCPMCR and LMPM

The service primitives including their associated parameters issued by DCPMCR received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.3.2.2.4.2 State machine description

The state machine is waiting for DCP-IdentifyReqPDUs in the OPEN state. The filter list shall be passed to the DCP user to check the filter criteria. The state machine waits in the state WRSP a certain time calculated by means of the clients Response Delay Factor to send the response in case the DCP user has recognized a match with local data. Otherwise, nothing shall be sent.

Local variables of the DCPMCR

SXID
This local variable contains the Transaction ID XID used to Identify services uniquely.

4.3.2.2.4.3 DCPMCR state table

Table 48 contains the complete description of the DCPMCR state machine. A LMPM_A_Data.ind primitive will be accepted if Type is DCP-IdentifyReqPDU.

Table 48 – DCPMCR state table

#	Current State	Event /Condition =>Action	Next State
1	OPEN	**DCP_Identify.rsp(ListOfData)** => ignore	OPEN
2	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /DCP-Identify-ReqPDU && Successful comparison with ListOfFilters => ResponseDelay :=((SA[4]*256+SA[5]) mod ResponseDelayFactor) *10 ms StartTimer(ResponseDelay) DCP_Identify.ind(ListOfFilter)	WRSP
3	OPEN	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** /!DCP-Identify-ReqPDU => ignore	OPEN
4	OPEN	**LMPM_A_Data.cnf (CREP, LMPM_status)** => ignore	OPEN
5	WRSP	**Timeout** /Stored A_SDU => A_SDU=Stored A_SDU LMPM_A_Data.req (CREP, DA, SA, Prio, A_SDU)	OPEN
6	WRSP	**Timeout** /!Stored A_SDU =>	OPEN
7	WRSP	**DCP_Identify.rsp(ListOfData)** => DCP_Header..op:= IDENT DCP_Header.xid:= SXID A_SDU := List of Data Store A_SDU	WRSP
8	WRSP	**LMPM_A_Data.ind (CREP, DA, SA, Prio, A_SDU)** => ignore	WRSP
9	WRSP	**LMPM_A_Data.cnf (CREP, LMPM_status)** => ignore	WRSP

4.4 Precision time control

4.4.1 FAL syntax description

4.4.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.4.1.2 APDU abstract syntax

Table 49 defines the abstract syntax of the PTCP PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 49 – PTCP APDU syntax

APDU name	APDU structure
PTCP-PDU	PTCP-Header, PTCP-RTASyncPDU ^ PTCP-FollowUpPDU ^ PTCP-AnnouncePDU ^ PTCP-RTCSyncPDU ^ PTCP-DelayReqPDU ^ PTCP-DelayResPDU ^ PTCP-DelayFuResPDU
PTCP-RTASyncPDU	PTCPSubdomain, PTCPTime, PTCPTimeExtension, PTCPMaster, [PTCPOption], PTCPEnd
PTCP-RTCSyncPDU	PTCPSubdomain, PTCPTime, PTCPTimeExtension, PTCPMaster, [PTCPRTData], [PTCPOption], PTCPEnd, APDU_Status
PTCP-AnnouncePDU	PTCPSubdomain, PTCPMaster, [PTCPOption], PTCPEnd
PTCP-FollowUpPDU	PTCPSubdomain, PTCPTime, [PTCPOption], PTCPEnd
PTCP-DelayReqPDU	PTCPSubdomain, PTCPDelayParameter, [PTCPOption], PTCPEnd
PTCP-DelayResPDU	PTCPSubdomain, PTCPDelayParameter, PTCPPortParameter, PTCPPortTime [a], [PTCPOption], PTCPEnd
PTCP-DelayFuResPDU	PTCPDelayParameter, [PTCPOption], PTCPEnd
[a] Each delay measurement requestor shall accept PTCP-DelayResPDU without this TLV.	

Table 50 defines structures for substitutions of elements of the APDU structures.

Table 50 – PTCP substitutions

Substitution name	Structure
PTCP-Header	PTCP_reserved_1, PTCP_reserved_2, PTCP_Delay10ns, PTCP_SequenceID, PTCP_Delay1ns, Padding [a], PTCP_CumulativeFrequencyOffset
PTCPSubdomain	PTCP_TLVHeader, PTCP_MasterSourceAddress, PTCP_SubdomainUUID
PTCPTime	PTCP_TLVHeader, [Padding*] [a], PTCP_Seconds, PTCP_NanoSeconds
PTCPTimeExtension	PTCP_TLVHeader, PTCP_Flags, PTCP_EpochNumber, PTCP_CurrentUTCOffset
PTCPMaster	PTCP_TLVHeader, PTCP_ClockVariance, PTCP_ClockUUID, PTCP_ClockStratum, PTCP_ClockRole, [Padding*] [a]
PTCPDelayParameter	PTCP_TLVHeader, PTCP_RequestSourceAddress, PTCP_RequestPortID, PTCP_SyncID, [Padding*] [a]
PTCPPortParameter	PTCP_TLVHeader, [Padding*] [a], PTCP_T2PortRxDelay, PTCP_T3PortTxDelay
PTCPPortTime	PTCP_TLVHeader, [Padding*] [a], PTCP_T2TimeStamp
PTCPOption	PTCP_TLVHeader, PTCP_OUI, PTCP_SubType, Data*, [Padding*] [a]
PTCPRTData	PTCP_TLVHeader(=0x7F, 22), PTCP_OUI(=00-0E-CF), PTCP_SubType(=1), [Padding*][a], IRDataUUID
PTCPEnd	PTCP_TLVHeader(0)
[a] 32bit alignment shall be ensured.	

4.4.1.3 Coding section related to common basic fields

4.4.1.3.1 Coding of the field PTCP_reserved_1

This field shall be coded as data type Unsigned32.

4.4.1.3.2 Coding of the field PTCP_reserved_2

This field shall be coded as data type Unsigned32.

4.4.1.3.3 Coding of the field PTCP_TLVHeader

The coding of this field shall be according to 3.7.3.4. and the individual bits shall have the following meaning:

Bit 0 – 8: PTCP_TLVHeader.Length
The value contains the sum of subsequent octets of the according block.

Bit 9 – 15: PTCP_TLVHeader.Type
This field shall be coded with the values according to Table 51.

Table 51 – PTCP_TLVHeader.Type

Value (hexadecimal)	Meaning	Use
0x00	PTCPEnd	Mandatory
0x01	PTCPSubdomain	Mandatory
0x02	PTCPTime	Mandatory
0x03	PTCPTimeExtension	Mandatory
0x04	PTCPMaster	Mandatory
0x05	PTCPPortParameter	Mandatory
0x06	PTCPDelayParameter	Mandatory
0x07	PTCPPortTime	Mandatory
0x08 – 0x7E	Reserved	
0x7F	OrganizationallySpecific	Optional

4.4.1.3.4 Coding section related to PTCP-Header

These fields contains the propagation delay time in nanoseconds. It is parted into a field for the 10 ns part and a field for the 1 ns part. All delay fields are only relevant for Sync-, FollowUp-, DelayRes- without DelayFuRes- and DelayFuRes-Messages.

4.4.1.3.4.1 Coding of the field PTCP_Delay10ns

This field shall be coded as data type Unsigned32 and contains the 10 ns part of the propagation delay time. This field shall be coded with the values according to Table 52.

Table 52 – PTCP_Delay10ns

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFE	Valid 10 nanosecond part of the propagation delay time
0xFFFFFFFF	Invalid This value indicates an overflow and marks the delay fields as invalid

4.4.1.3.4.2 Coding of the field PTCP_Delay1ns

The field contains the 1 ns part and the sign of the propagation delay time.

The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0 – 3: PTCP_Delay1ns.Value
The value contains the 1 ns part of the propagation delay time. This field shall be coded with the values according to Table 53.

Table 53 – PTCP_Delay1ns.Value

Value (hexadecimal)	Meaning
0x00 – 0x09	Number of ns
0x0A – 0x0F	Reserved

Bit 4 – 6: PTCP_Delay1ns.reserved
This field shall be set according to 3.7.3.2.

Bit 7: PTCP_Delay1ns.Sign
The value contains the sign of the propagation delay time. This field shall be coded with the values according to Table 54.

Table 54 – PTCP_Delay1ns.Sign

Value (hexadecimal)	Meaning
0x00	Treat PTCP_Delay1ns.Value and PTCP_Delay10ns as positive value
0x01	Treat PTCP_Delay1ns.Value and PTCP_Delay10ns as negative value

4.4.1.3.4.3 Coding of the field PTCP_CumulativeFrequencyOffset

This field shall be coded as data type Integer32 with the values according to Table 55. It contains the cumulated frequency offset of the transmission path. Every node which forwards the PTCP-RTASyncPDU shall add its scaled frequency offset according to Equation (5).

$$\text{PTCP\_CumulativeFrequencyOffset} = \sum_{i=0}^{n} \text{ScaledFrequencyOffset}[i] \quad (5)$$

$$\text{RCF} = 1 + \text{PTCP\_CumulativeFrequencyOffset} \times 2^{-30} \quad (6)$$

Table 55 – PTCP_CumulativeFrequencyOffset

Value (decimal)	Value (hexadecimal)	Meaning
214 749 – 2 147 483 647		Positive value
0 – 214 748		Positive value 0 is equal to zero ppm frequency offset 214748 is equal to 200 ppm frequency offset
	0x80000000	Invalid The PTCP_CumulativeFrequencyOffset shall no be used and the Equation (5) shall not apply. If no valid scaled frequency offset exists, the PTCP_CumulativeFrequencyOffset shall be set to invalid
-1 – -214 748		Negative value 0 is equal to zero ppm frequency offset -214748 is equal to -200 ppm frequency offset
-214 749 – -2 147 483 647		Negative value

4.4.1.3.4.4 Coding of the field PTCP_SequenceID

This field shall be coded as data type Unsigned16 with the values according to Table 56.

Table 56 – PTCP_SequenceID

Meaning	Usage
Sync frame	The PTCP master increments these field for every sync frame.
Follow up frame	The PTCP master or a follow up injecting PTCP forwarder mirrors the value from the Sync frame.
Delay request frame	The PTCP delay messurement requester increments these field for every messurement.
Delay response frame	The responder mirrors the value from the Delay request frame.
Delay response follow up frame	The responder uses the value from the Delay response frame.

4.4.1.3.5 Coding of the field PTCP_OUI

This field shall be coded as OctetString[3] with the Organizationally unique identifier (OUI) as defined in IEEE 802.

Note The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <http://standard.ieee.org/regauth>

4.4.1.3.6 Coding of the field PTCP_SubType

This field shall be coded as Unsigned8. The value is organizationally specific. For the OUI the parameter PTCP_SubType shall be coded as in Table 57.

Table 57 – PTCP_SubType for OUI (=00-0E-CF)

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	
0x01	IRDATA	Used for RT_CLASS_3
0x02 – 0xFF	Reserved	

4.4.1.3.7 Coding of the field PTCP_Seconds

This field shall be coded as data type Unsigned32. The value contains the time (resolution of 1 s).

4.4.1.3.8 Coding of the field PTCP_NanoSeconds

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 58.

Table 58 – PTCP_NanoSeconds

Value (hexadecimal)	Meaning
0x00000000 – 0x3B9AC9FF	Delay in ns
0x3B9ACA00 – 0xFFFFFFFF	Reserved

4.4.1.3.9 Coding of the field IRDataUUID

This field shall be coded as data type UUID.

4.4.1.4 Coding section of PTCP-PDU specific fields

4.4.1.4.1 Coding of the field PTCP_MasterSourceAddress

This field shall be coded as data type OctetString[6]. The value of the field Destination Address shall be according to IEEE 802 MAC address.

4.4.1.4.2 Coding of the field PTCP_SubdomainUUID

This field shall be coded as data type UUID.

4.4.1.4.3 Coding of the field PTCP_Flags

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: PTCP_Flags.reserved_1
This field shall be set according to 3.7.3.2.

Bit 8 – 9: PTCP_Flags.LeapSecond
This field shall be set according Table 59.

Table 59 – PTCP_Flags.LeapSecond

Value (hexadecimal)	Meaning
0x00	No leap second
0x01	Last minute has 61 s
0x02	Last minute has 59 s
0x03	Reserved

Bit 10 – 15: PTCP_Flags.reserved_2
This field shall be set to zero.

4.4.1.4.4 Coding of the field PTCP_EpochNumber

This field shall be coded as data type Unsigned16. The value of the PTCP_EpochNumber field shall be the value of epoch_number of the global time properties data set of the clock issuing this message according to Table 60 and Table 61.

Table 60 – Timescale correspondence between MJD, UTC, and PTCP_EpochNumber

MJD (Modified Julian Day)	UTC (Universal Time Coordinated)	UTC Offset (Leap seconds)	Value PTCP_Seconds	Value PTCP_Epoch-Number
0:00:00 15 020	1 January 1900 – 00:00:00	0		0
0:00:00 40 587	1 January 1970 – 00:00:00		0	0
0:00:00 41 317	1 January 1972 – 00:00:00	10	63 072 000 + 10	0
16:57:44 51 357	28 June 1999 – 16:57:44	32	930 589 064 + 32	0
	1 January 2006 – 00:00:00	33		0

Table 61 – Timescale correspondence between PTCP_EpochNumber, PTCP_Second, PTCP_Nanosecond, CycleCounter, and SendClockFactor

CycleCounter value (hexadecimal)	PTCP_EpochNumber value (decimal)	PTCP_Second value (decimal)	PTCP_Nanosecond value (decimal)	SendClockFactor value (decimal)
0x0000000000000000	0	0	0	32
0x0000000000010000	0	65	536 000 000	32
0x0000000100000000	0	4 294 967	296 000 000	32
0x0001000000000000	65	2 302 102 470	656 000 000	32

$$\text{PTCP\_Epoch} = \text{PTCP\_EpochNumber} \times 0x100000000 \quad (7)$$

$$\text{PTCP\_Time} = \text{PTCP\_Epoch} + \text{PTCP\_Second} + \text{PTCP\_Nanosecond} \quad (8)$$

$$\text{CycleCounter} = \frac{\text{PTCP\_Time}}{\text{SendClockFactor} \times 31{,}25\mu s} \quad (9)$$

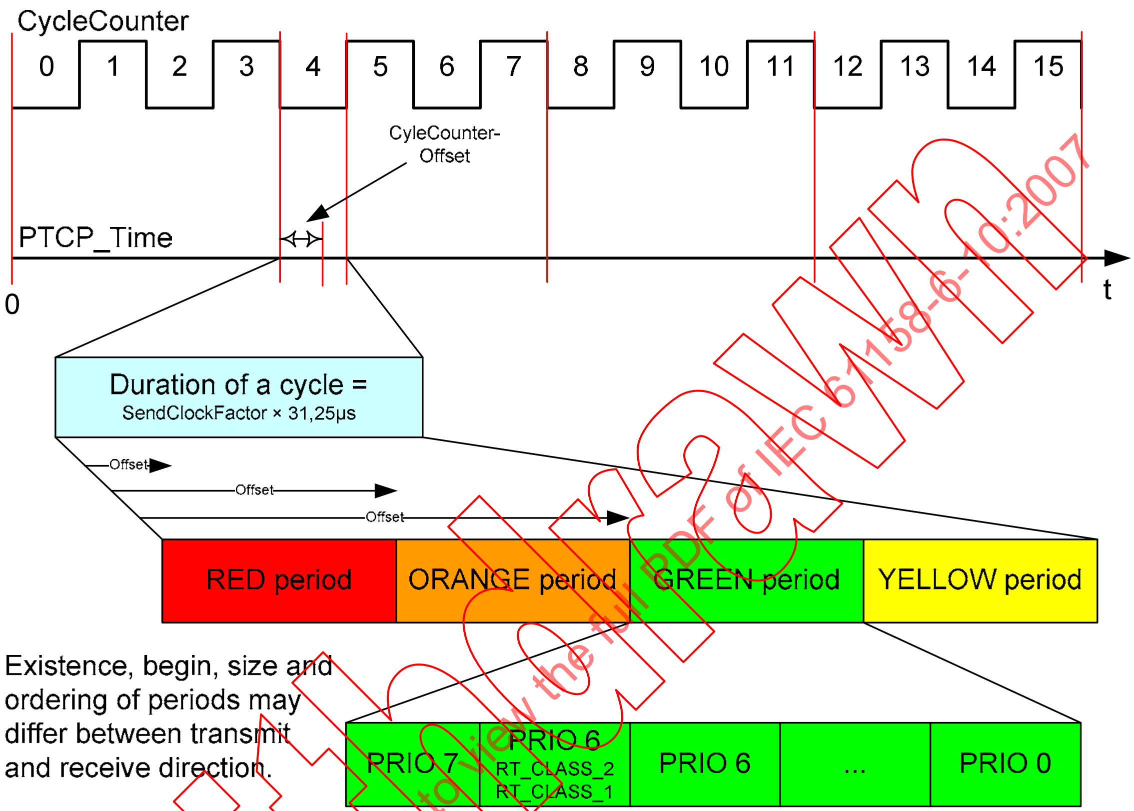

Figure 13 – Timescale correspondence between PTCP_Time and CycleCounter

For the synchronisation of the CycleCounter as shown in Figure 13, the calculation shall accept PTCP_Time values with an offset to the cycle begin.

CycleCounter = PTCP_Time div (SendClockFactor × 31,25 µs) (10)

CycleCounterOffset = PTCP_Time mod (SendClockFactor × 31,25 µs) (11)

4.4.1.4.5 Coding of the field PTCP_CurrentUTCOffset

This field shall be coded as data type Integer16. The value of the current PTCP_CurrentUTCOffset field shall be the value of current_utc_offset of the global time properties data set of the clock issuing this message according to Table 60.

4.4.1.4.6 Coding of the field PTCP_ClockUUID

This field shall be coded as data type UUID.

4.4.1.4.7 Coding of the field PTCP_ClockStratum

This field shall be coded as Unsigned8. The clock stratum, or stratum number, describes one measure of the quality of a clock and shall be coded as in Table 62.

Table 62 – PTCP_ClockStratum

Value (hexadecimal)	Meaning	Usage
0x00	Force	May be used temporarily for special purposes by PTCP implementations to force a clock to be deemed better than other clocks in the system.
0x01	Primary	Designates the clock as a primary reference standard traceable to a recognized standard source of time. Note GPS clocks, calibrated atomic clocks, etc. fall into this stratum A stratum 1 clock shall not be synchronized using the PTCP protocol to another clock in a PTCP system.
0x02	Secondary	Designates the clock as a secondary standard reference clock. The clock shall be: Directly (not via PTCP) synchronized to a stratum 1 clock or another source deemed to be a correct source of time for the PTCP subdomain or previously directly synchronized to a stratum 1 clock or another source deemed to be a correct source of time for the PTCP subdomain.
0x03	TimingSignal	The lowest possible stratum value if not 1 or 2 for a clock that is capable of issuing external timing signals.
0x04 – 0xFF	Reserved	

4.4.1.4.8 Coding of the field PTCP_ClockVariance

This field shall be coded as data type Integer16. The value characterizes the quality of a clock.

4.4.1.4.9 Coding of the field PTCP_ClockRole

This field shall be coded as Unsigned8 according to Table 63.

Table 63 – PTCP_ClockRole

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	
0x01	Primary	Primary PTCP-Master
0x02	Secondary	Secondary PTCP-Master
0x03 – 0xFF	Reserved	

4.4.1.4.10 Coding of the field PTCP_T2PortRxDelay

This field shall be coded as data type Unsigned32. The value contains the port-receive-delay in 1 ns.

4.4.1.4.11 Coding of the field PTCP_T3PortTxDelay

This field shall be coded as data type Unsigned32. The value contains the port-transmit-delay in 1 ns.

4.4.1.4.12 Coding of the field PTCP_RequestSourceAddress

This field shall be coded as data type OctetString[6]. The value of the field PTCP_RequestSourceAddress shall be according to IEEE 802 MAC address.

4.4.1.5 Coding of the field PTCP_RequestPortID

This field shall be coded as data type Unsigned8 with the values according to Table 64.

Table 64 – PTCP_RequestPortID

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0xFF	Port number

4.4.1.5.1 Coding of the field PTCP_SyncID

This field shall be coded as data type Unsigned8 with the values according to Table 65.

Table 65 – PTCP_SyncID

Value (hexadecimal)	Meaning	Use
0x00	Clock	Send Clock and phase synchronization
0x01	Time	Time synchronization
0x02 – 0xFF	Reserved	

4.4.1.5.2 Coding of the field PTCP_T2TimeStamp

This field shall be coded as data type Unsigned32 with the values according to Table 66.

Table 66 – PTCP_T2TimeStamp

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	TimeStamp in nanoseconds

4.4.2 AP-Context state machine

There is no AP-Context State Machine defined for this Protocol.

4.4.3 FAL Service Protocol Machines (FSPMs)

There is no FAL Service Protocol Machine defined for this Protocol.

4.4.4 Application Relationship Protocol Machines (ARPMs)

4.4.4.1 Generic Synchronization with PTCP

4.4.4.1.1 Timestamp

For each received and transmitted sync, delay request or delay response message a time stamping unit has to generate a time stamp. The PTCP message timestamp point shall correspond to the leading edge of the first bit of the octet immediately following the Start Frame Delimiter octet of an IEEE 802.3 frame as shown in Figure 14.

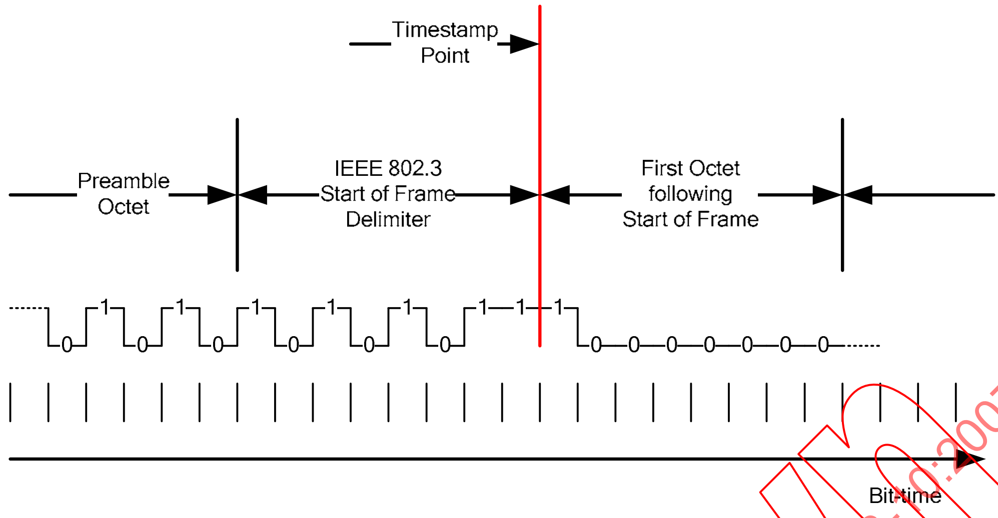

Figure 14 – Message timestamp point

Figure 15 shows the four timestamps used for synchronization.

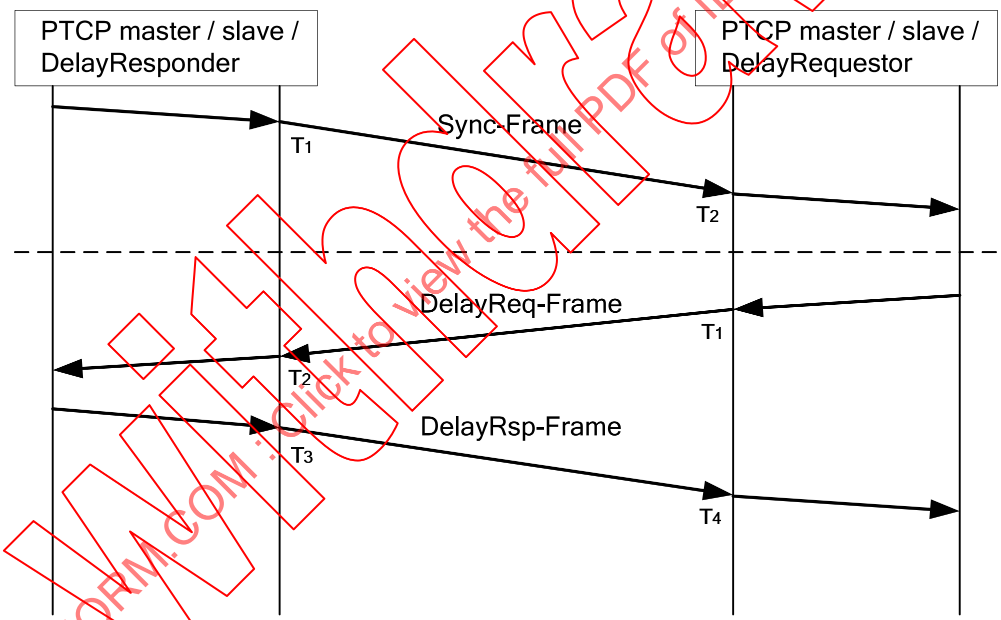

Figure 15 – Four message timestamps

SendTimeStamp T1
is measured by the local clock of the port which sends a Sync- or a DelayReq-Frame.

ReceiptTimeStamp T2
is measured by the local clock of the port which receives a Sync- or a DelayReq-Frame.

SendTimeStamp T3
is measured by the local clock of the port which sends a DelayRes-Frame.

ReceiptTimeStamp T4
is measured by the local clock of the port which receives a DelayRes-Frame.

4.4.4.1.2 Line delay measurement and peer rate compensation

The line delay measurement and the peer rate compensation calculation between DelayRequestor and DelayResponder are described in Figure 16. DelayReq-, DelayRes and DelayFuRes-Messages shall be used for delay measurement and shall not be propagated. The DelayFuRes-Frame is used to transmit the value of ResDelay to the DelayRequestor.

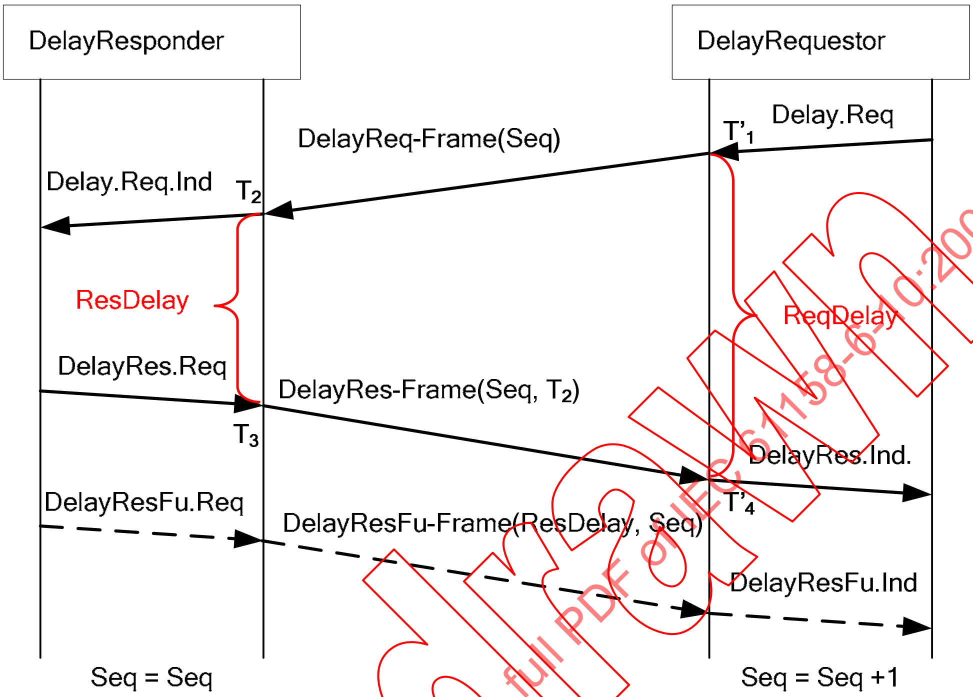

Figure 16 – Line delay protocol with follow up

If a time stamping unit is capable to calculate the response delay time on the fly and insert the response time in the delay response frame no delay follow up message is necessary.

The line delay measurement whitout DelayFuRes-message between DelayRequestor and DelayResponder is described in Figure 17.

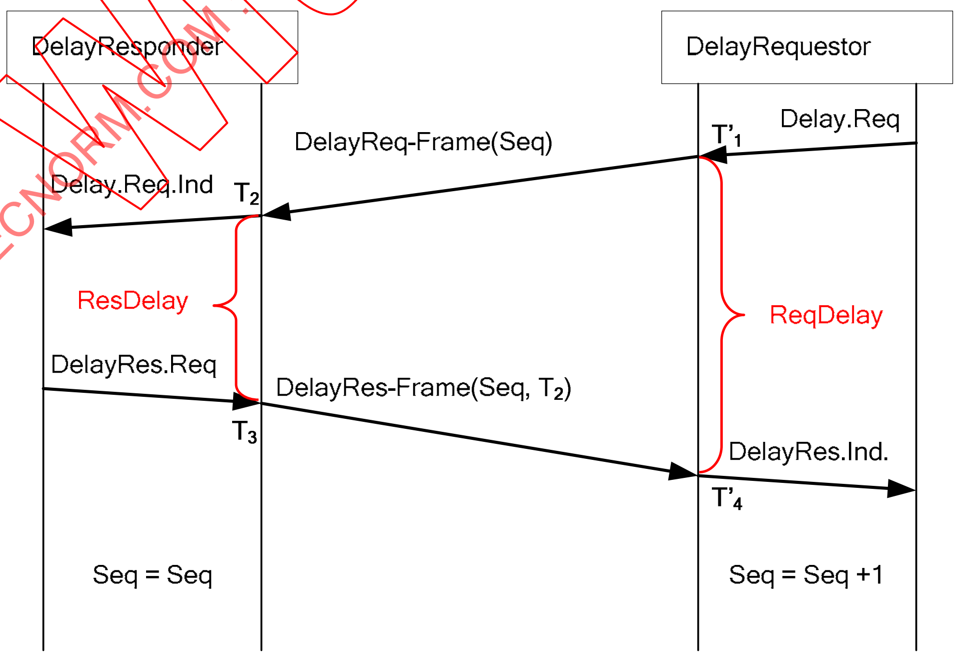

Figure 17 – Line delay protocol without follow up

The calculation of line delay and peer rate compensation factor is shown in following equations. If the DelayRes-PDU contains the field T_2, the Equation (14) shall be used, otherwise Equation (15) shall be used.

$$ResDelay = T_3 - T_2 \quad (12)$$

$$ReqDelay = T'_4 - T'_1 \quad (13)$$

$$RCF_{peer} = \frac{T_2(Seq) - T_2(Seq-1)}{T'_1(Seq) - T'_1(Seq-1)} \quad (14)$$

$$RCF_{peer} = 1 \quad (15)$$

$$ResDelay_{peer} = \frac{ResDelay}{RCF_{peer}} \quad (16)$$

$$LineDelay_{peer} = \frac{ReqDelay - ResDelay_{peer}}{2} \quad (17)$$

$$RCF_{Scal} = (RCF_{peer} - 1) \times 2^{30} \quad (18)$$

$$RCF_{Trunc} = Truncate\_to\_Integer32(RCF_{Scal}) \quad (19)$$

$$ScaledFrequencyOffset = RCF_{Trunc} \quad (20)$$

The delay measurement shall be repeated in cycles.

4.4.4.1.3 Line delay calculation

The line delay between two devices supporting PTCP is determined as described in Figure 18.

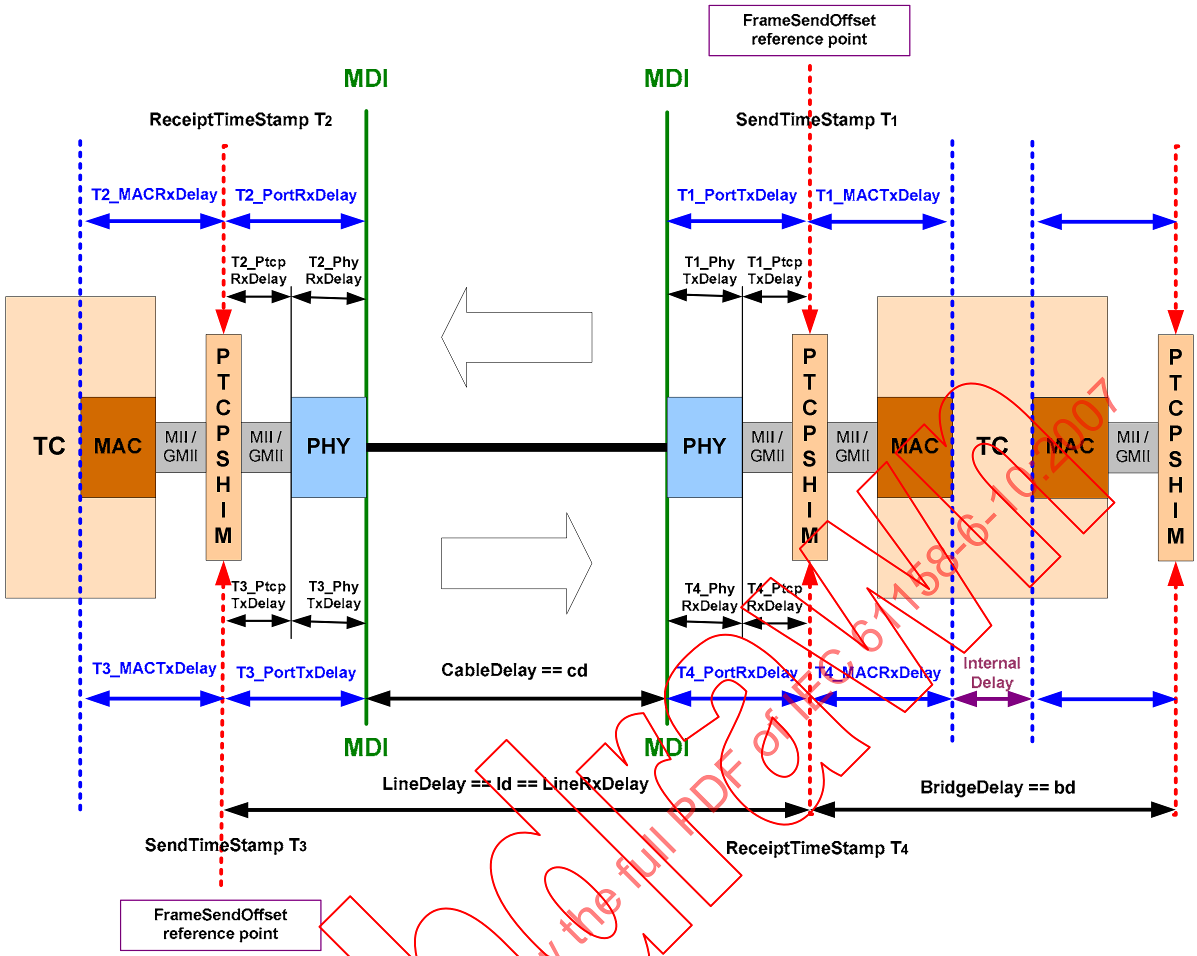

Figure 18 – Line delay measurement

NOTE 1 The Reduced Media Independent Interface (RMII) synchronisates its internal 50MHz clock with the 25 MHz clock of the PHY. It results in a quantization error of 20 ns. This jitter can not be compensated. The Media Independent Interface (MII) is using the PHY clock without this error.

NOTE 2 The cable type Class D has a maximal delay skew of 45 ns for 100 m cable length. It results in a quantization error of 45 ns. This jitter can not be compensated. A selection of better cable reduces this error.

A port receive/transmit-delay (example: T1_PortTxDelay) contains the line delay of the used PHY component and the deviation from the reference time stamp.

T1_PortTxDelay = T1_PhyTxDelay + T1_PtcpTxDelay (21)

T3_PortTxDelay = T3_PhyTxDelay + T3_PtcpTxDelay (22)

T2_PortRxDelay = T2_PhyRxDelay + T2_PtcpRxDelay (23)

T4_PortRxDelay = T4_PhyRxDelay + T4_PtcpRxDelay (24)

CableDelay = (ReqDelay – ResDelay – T1_PortTxDelay – T2_PortRxDelay –
T3_PortTxDelay – T4_PortRxDelay) / 2 (25)

LineDelay = CableDelay + T3_PortTxDelay + T4_PortRxDelay (26)

LineSyncDelay is the line delay of the Sync-Frame. Due to the asymmetry of the involved communication path and due to the resolution of the time stamps it is basically impossible to determine the line delay exactly.

A PTCP clock can detect non PTCP neighbors by checking the line delay between nodes. As a switch delay is in the range of several (>2) us, the delay between two nodes connected with 100 m Twisted Pair cabling (100 Base TX) is about 500 ns.

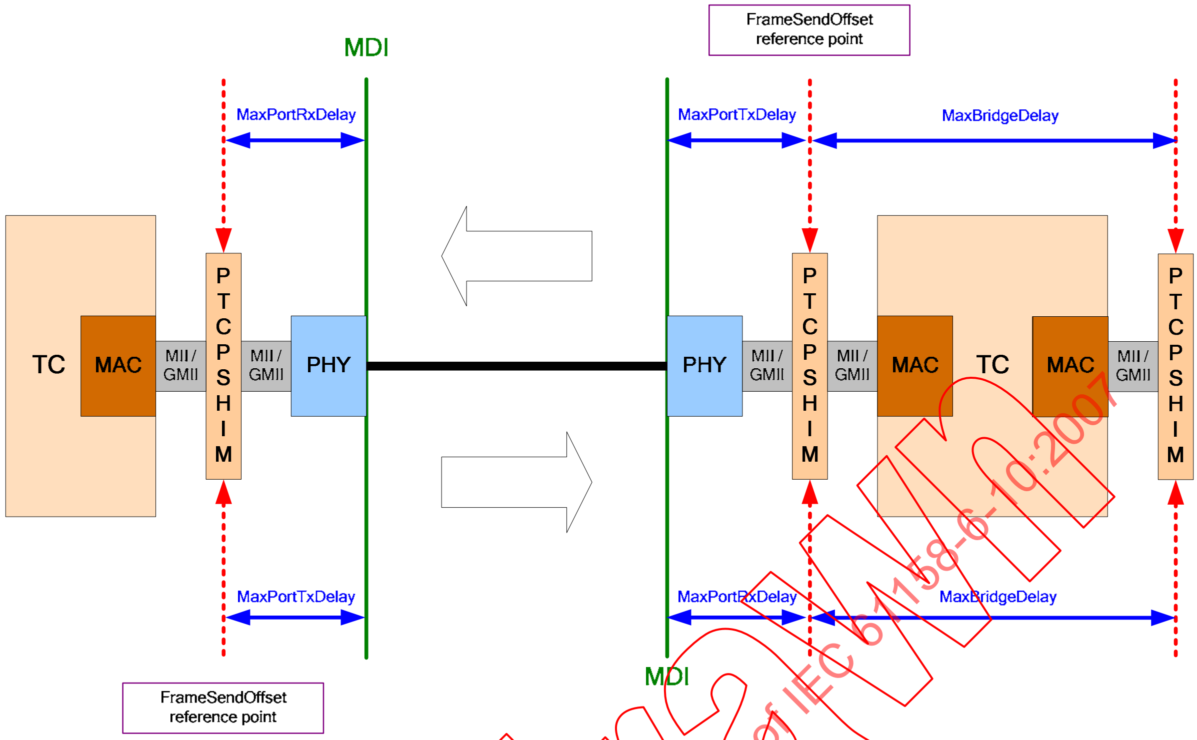

Figure 19 – Model parameter for GSDML usage

Figure 19 shows the model parameter for GSDML usage. The parameter MaxBridgeDelay, MaxPortTXDelay and MaxPortRXDelay offers engineering tools a calculation basis for RT_CLASS_3.

4.4.4.1.4 Sync-Frame forwarding

A device which forwards PTCP sync messages measures the bridge delay as shown in Figure 20 and adds this time to the delay field in the sync or follow up message.

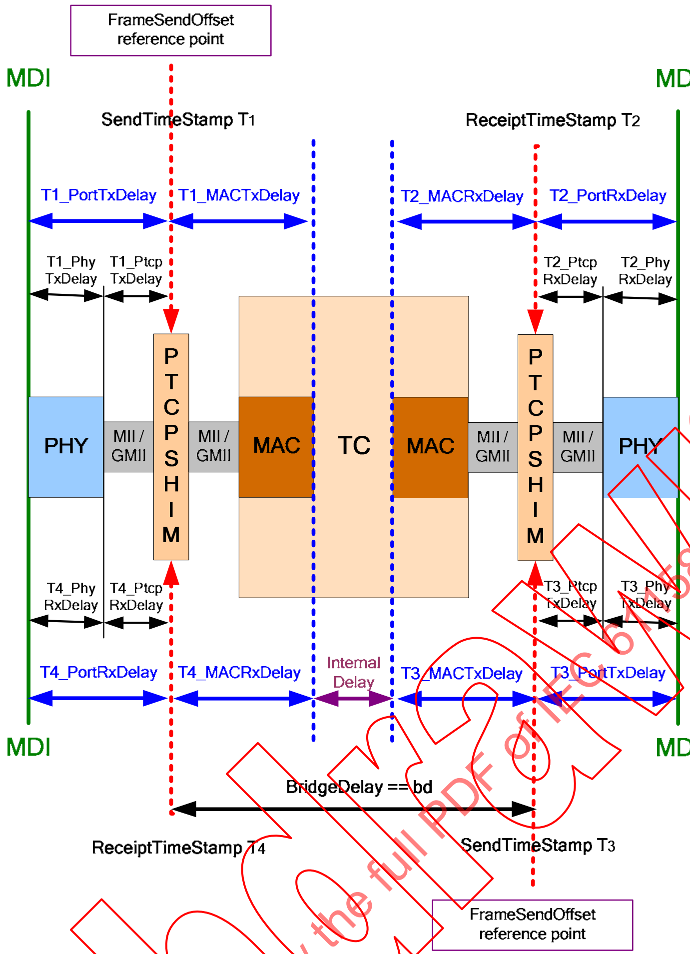

Figure 20 – Bridge delay measurement

The calculation of the bridge delay is shown in following equations.

$$BridgeDelay = T_3 - T_4 \tag{27}$$

$$BridgeDelay_{SyncMaster} = BridgeDelay \times RCF_{SyncMaster} \tag{28}$$

The principle of synchronization through a sequence of devices can be seen in Figure 21. The node emitting the time frames for synchronization in the network is known as PTCP master. All other nodes that receive and/or pass on these frames are known as PTCP slave. A PTCP master uses a reference to a local time system or a global time source.

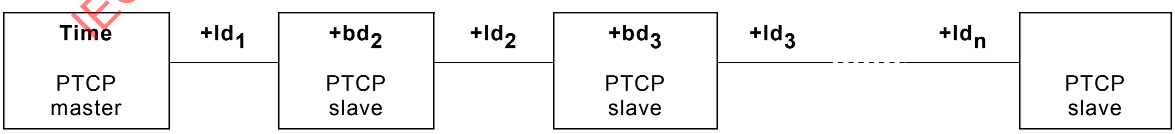

Figure 21 – Delay accumulation

The Sync-Frames are forwarded as peer to peer frames. Every device has to adjusted the Sync-Frame by

- adding its bridge delay bd_x (bridge delay measurement is shown in Figure 20),
- adding the line delay ld_x (line delay measurement is shown in Figure 18),

to the delay field of the Sync-Frame as it passes through. Each PTCP slave knows the propagation delay of the sync frame. When the time of arrival T_2 is known, the drift compared to the PTCP master can also be determined.

4.4.4.1.5 Rate compensation

4.4.4.1.5.1 Bridge delay

If the bridge delay is significant (several 100 µs), then the values shall be corrected by rate compensation in order to achieve high precision synchronization.

EXAMPLE If a frame of 12 000 Bit times will be send out the delay of a subsequent time frame is about 120 µs at fast Ethernet speed. If there is a clock drift of 100 PPM and the PTCP master has a clock drift of -100 PPM this would result in an error of 24 ns. If there is no priority for PTCP frames and there are 10 frames queued, this could result in an additional error of 240 ns which may be inacceptable. That means that a large delay value means less precicision of the delay value. Each clock sampling at a node is always associated with a synchronization jitter. In addition to this, there is always control inaccuracy for each bridge. The maximum time difference between any two nodes of a network is dependent on the topology – cascading of bridges result in an inaccuracy depending on the level of cascading.

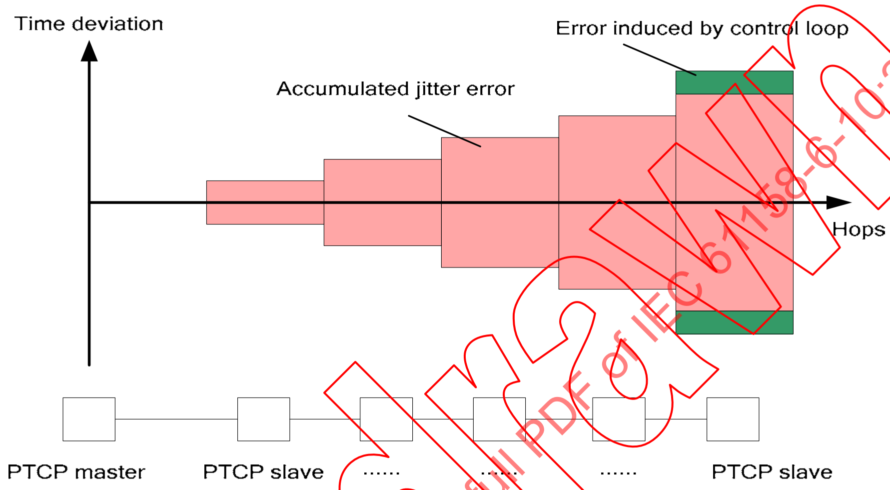

Figure 22 – Worst case accumulated time deviation of synchronization

The highest required accuracy for synchronization applies only to two adjacent nodes.

4.4.4.1.5.2 Line delay measurement

If the ReqDelay shown in Figure 17 is significant (several 100 us), then the values of ReqDelay and ResDelay shall be corrected by rate compensation in order to achieve high precision synchronization.

4.4.4.1.6 Time deviation measurement

To achieve the measurement of the deviation hardware signaling is necessary. For every supported PTCP_SyncID a signal shall be generated. It may only be accessible in test lab environment.

Figure 23 and Figure 24 shows in principle the measurement of the deviation.

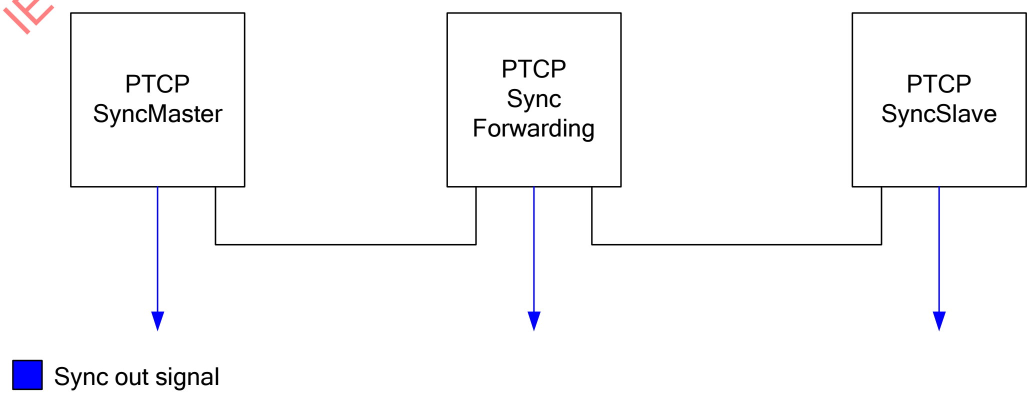

Figure 23 – Scheme for measurement of deviation

Every cycle as shown in Figure 13 shall be signaled for clock synchronization. Every second shall be signaled for time synchronization.

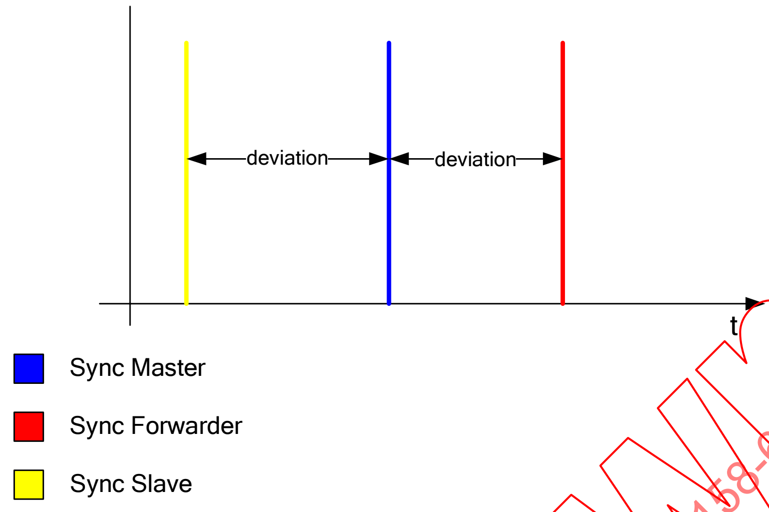

Figure 24 – Measurement of deviation

4.4.4.1.7 Synchronization Protocol

4.4.4.1.7.1 Synchronization with Sync-Frame

PTCP masters are sending Sync-Frames periodically. The synchronization scheme as described in Figure 25 requires transferring the exact sending time of the Sync-Frame as initial value of the delay.

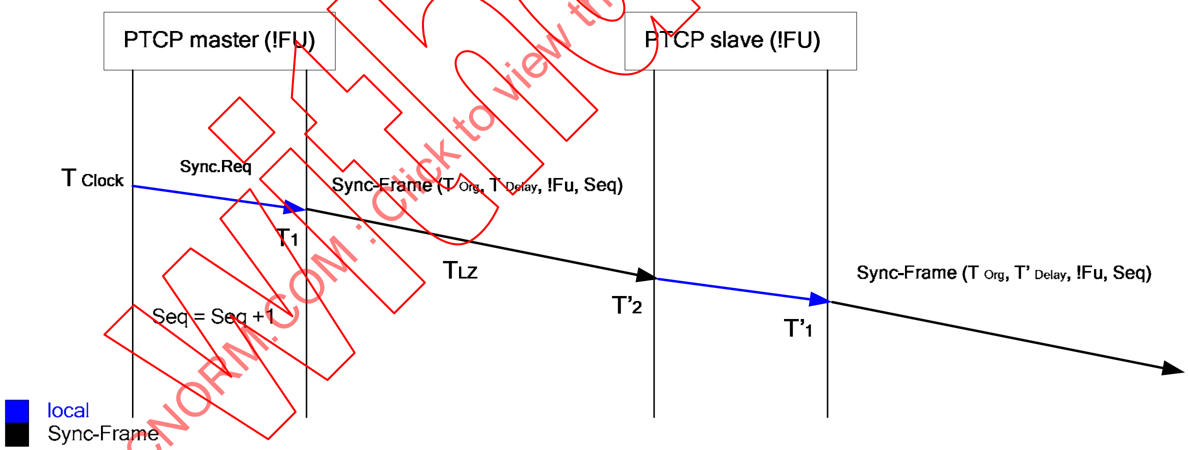

Figure 25 – Sending Sync-Frame without Follow Up-Frame

$$T_{Org} = T_{Clock} \tag{29}$$

$$T_{Delay} = T_1 – T_{Clock} \tag{30}$$

$$T'_{err} = T'_2 – (T_{Org} + T_{Delay} + T_{LZ}) \tag{31}$$

$$T'_{Delay} = T_{Delay} + T_{LZ} + (T'_1 – T'_2) \tag{32}$$

4.4.4.1.7.2 Synchronization with Sync- and FollowUp-Frame

Figure 26 shows a PTCP master which can not modify a frame before transmission. A PTCP master who is not able to put the exact sending time in a Sync-Frame will use a FollowUp-Frame.

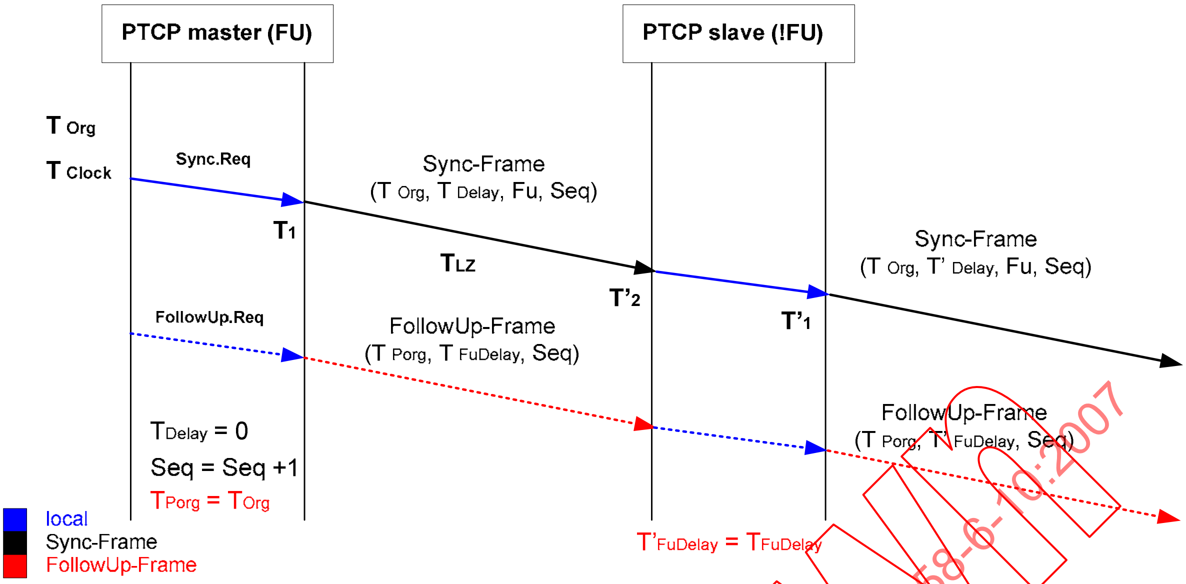

Figure 26 – Sending Sync-Frame with FollowUp-Frame

The delay field in the Sync-Frame shall be set to 0. The delay ($T_{FuDelay}$) in the associated FollowUp-Frame contains the initial value of the delay.

$$T_{Org} = T_{Clock} \quad (33)$$

$$T_{FuDelay} = T_1 - T_{Clock} \quad (34)$$

$$T_{err} = T'_2 - (T_{Porg} + T'_{Delay} + T'_{FuDelay} + T'_{LZ}) \quad (35)$$

$$T'_{Delay} = T_{Delay} + T_{LZ} + (T'_1 - T'_2) \quad (36)$$

A PTCP slave as shown in Figure 27 can modify a frame before transmission (!FU-Node) adds its internal bridge delay and the line delay to the delay value in the Sync-Frame as it passes through. A FollwUp-Frame is forwarded without modifications.

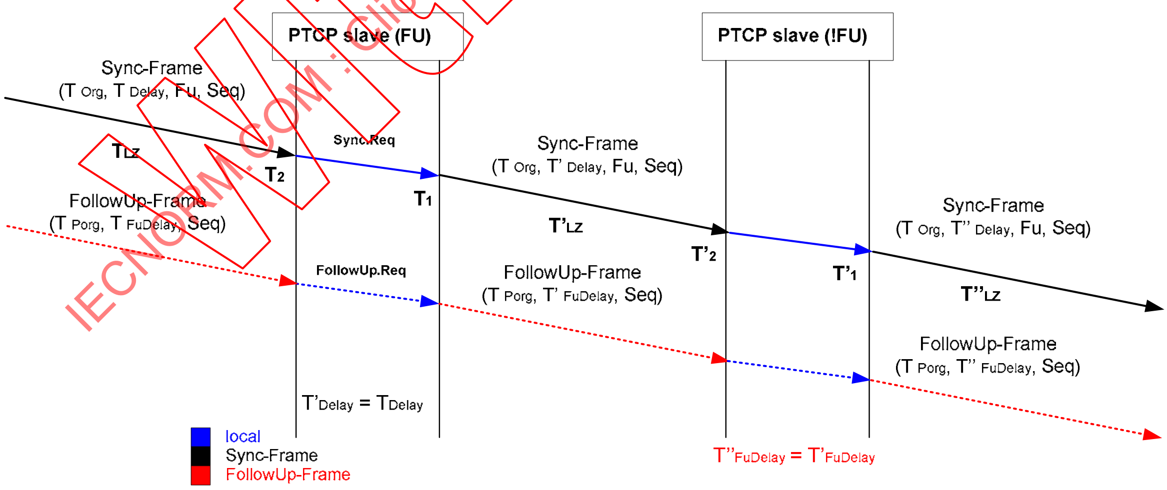

Figure 27 – Forwarding Sync- and FollowUp-Frame

$$T_{err} = T_2 - (T_{Porg} + T_{Delay} + T_{FuDelay} + T_{LZ}) \quad (37)$$

$$T'_{FuDelay} = T_{FuDelay} + T_{LZ} + (T_1 - T_2) \quad (38)$$

$$T'_{err} = T'_2 - (T_{Porg} + T'_{Delay} + T'_{FuDelay} + T'_{LZ}) \quad (39)$$

$$T''_{Delay} = T'_{Delay} + T'_{LZ} + (T'_1 - T'_2) \quad (40)$$

4.4.4.1.7.3 Concurrent operation of synchronization variants

Some implementations of IEEE 802.3 do not allow modifying a frame during transmission. The exact delay between T_{Org} and the time sending Sync-Frame will be sent in a second frame which is called FollowUp-Frame. A PTCP slave (FU-Node), which is not capable of adding its internal bridge delay and the line delay to the delay value in the Sync-Frame as it passes through, puts its bridge delay and the line delay to the delay value in a generated FollowUp-Frame. This principle is shown in Figure 28.

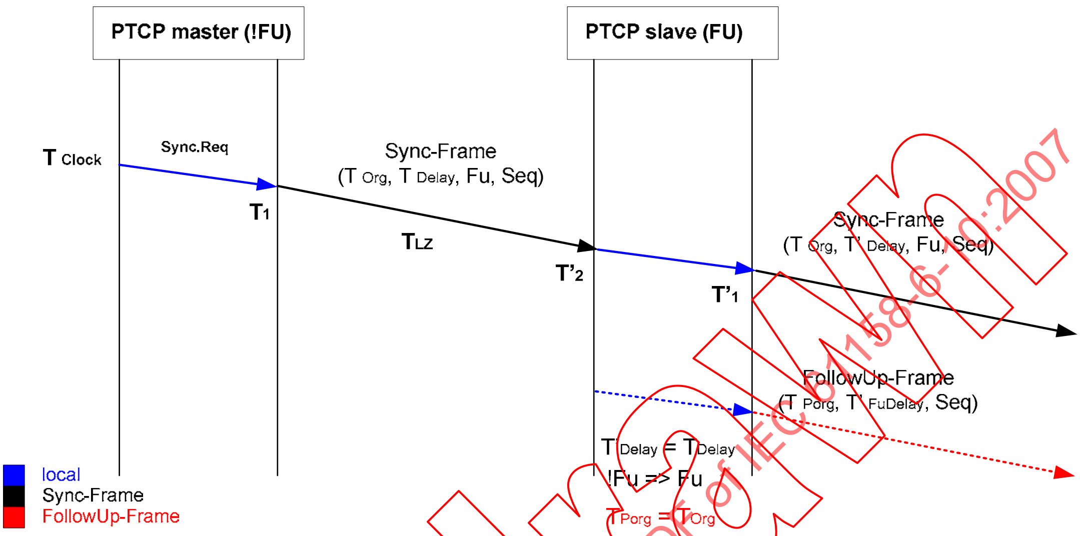

Figure 28 – Transition between Synchronization Variants

$$T_{Org} = T_{Clock} \tag{41}$$

$$T_{Offset} = T_2 - (T_{Org} + T_{Delay} + T_{LZ}) \tag{42}$$

$$T'_{FuDelay} = T_{LZ} + (T_1 - T_2) \tag{43}$$

$$T'_{err} = T'_2 - (T_{Porg} + T'_{Delay} + T'_{FuDelay} + T'_{LZ}) \tag{44}$$

$$T''_{Delay} = T'_{Delay} + T'_{LZ} + (T'_1 - T'_2) \tag{45}$$

4.4.4.1.8 Error recovery

A link down will delete the line delay. A line delay measurement error will stop the forwarding of all sync messages.

To deal with redundancy switchover in case of redundant paths, a Sync-Frame shall be forward only if the sequence number difference to the previous forwarded message is positive. This prevents duplicates which may corrupt the follow up sequence.

The use of an alternate path is possible as long as there is a valid line delay measurement available.

There are 2 timeouts:

- Delay measurement:
  Allowed time between delay request and delay response
- Monitoring of sync:
  Allowed time between two subsequent sync frames

The timeout of the line delay measurement shall be higher as the timeout of the time master and is calculated independently at every update of the line delay.

For resource limitations the size of the master data base can be limited. Two masters shall be possible at any time. This is possible as the BMA algorithm will cancel the activities of a Sync master.

The line delay values can be restricted for some technologies (e.g. 100 Base TX have delay values below 1 µs). Error shall be reported but the reaction is beyond the scope of this specification.

4.4.4.2 Line delay measurement

4.4.4.2.1 Line delay request Protocol Machine (DelayRequest)

4.4.4.2.1.1 Primitive definitions

4.4.4.2.1.1.1 Primitives exchanged between MSM and ASE

The service primitives including their associated parameters issued by DelayRequest user received by DelayRequest and vice versa are described in the PTCP ASE in the service definition.

4.4.4.2.1.2 State machine description

The line delay measurement shall be initiated by each port and should be repeated in intervals of 0.25, 0.5, 1, 2, 4, 8 or 16 seconds. The state machine which sends a delay request frame is called DelayRequestor. The line delay measurement shall be done with the smallest used SyncID.

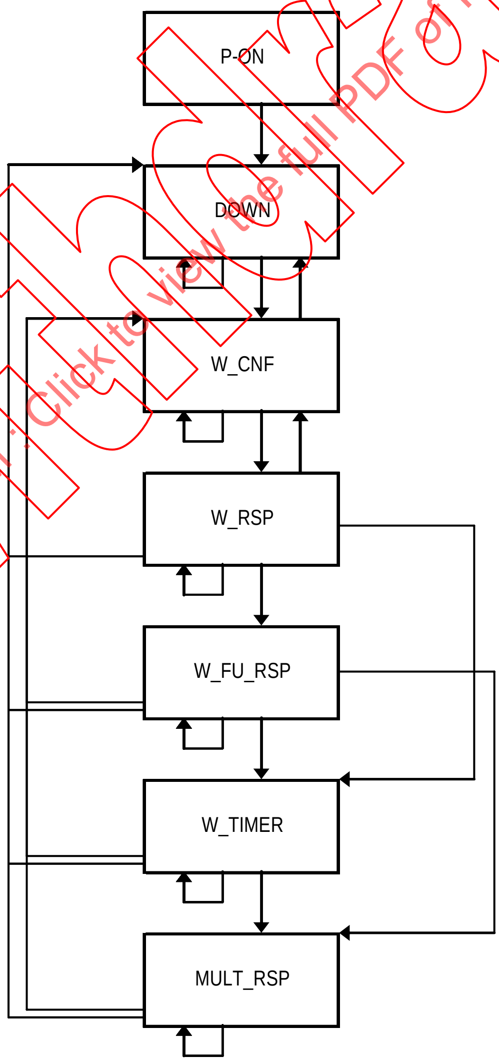

Figure 29 – State transition diagram of delay request

States of the **DelayRequest**

POWER_ON
Initialization of local data.

DOWN
Link of port is down. The line delay measurement shall be initiated after link up by sending a delay request frame.

W_CNF
Wait for confirmation of delay request and store the transmit timestamp of the delay request message.

W_RSP
Wait for delay response frame from the DelayResponder.

W_FU_RSP
Wait for delay response follow up frame and calculate line delay.

W_TIME
Wait for timeout to repeat the line delay measurement.

MULT_RSP
More delay response frames received. Wait for timeout and repeat the line delay measurement.

Local variables of the **DelayRquest**

Tx_TStamp_T1
This local variable contains the transmit timestamp of the delay request frame.

Rx_TStamp_T4
This local variable contains the recieve timestamp of the delay response frame.

SequenceID
This local variable contains the sequence number of the delay request message. It shall be incremented with every new delay request messages.

MultRsp
This local variable is set if multiples delay responses are received.

4.4.4.2.1.3 DelayRequest state table

Table 67 contains the state table used by DelayRequest.

Table 67 – DelayRequest state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> SequenceID := 0 ErrorCount := 0 Delay := 0 Tx_TStamp_T1 := NIL Rx_TStamp_T4 := NIL	DOWN
2	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID <> Port => ignore	DOWN
3	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type <> FULL_DUPLEX => ignore	DOWN
4	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port MAU_Type == FULL_DUPLEX && LINK_Status <> Up => ignore	DOWN
5	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up => ErrorCounter := 0 DelayReqT.start (DELAY_REQ_INTERVAL) DelayReq_Req (Port, SequenceID)	W_CNF
6	DOWN	**DelayReq_Cnf (D_Port, TStamp, Status)** => ignore	DOWN
7	DOWN	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	DOWN
8	DOWN	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	DOWN
9	W_CNF	**DelayReq_Cnf (D_Port, TStamp, Status)** /D_Port <> Port => ignore	W_CNF
10	W_CNF	**DelayReq_Cnf (D_Port, TStamp, Status)**/D_Port == Port&& Status <> OK=>DelayReqT.stopSequenceID++ErrorCount++DelayReqT.start (DELAY_REQ_INTERVAL)DelayReq_Req (Port, SequenceID)	W_CNF
11	W_CNF	**DelayReq_Cnf (D_Port, TStamp, Status)** /D_Port == Port && Status == OK => Tx_TStamp_T1 := TStamp	W_RSP
12	W_CNF	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	W_CNF
13	W_CNF	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	W_CNF
14	W_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID <> Port => ignore	W_CNF

#	Current State	Event /Condition =>Action	Next State
15	W_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type <> FULL_DUPLEX => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
16	W_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && LINK_Status == DOWN => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
17	W_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up => ignore	W_CNF
18	W_CNF	**DelayReqT.expired**/ErrorCount < MAX_COUNT_ERROR=>SequenceID++ErrorCount++DelayReqT.start (DELAY_REQ_INTERVAL)DelayReq_Req (Port, SequenceID)	W_CNF
19	W_CNF	**DelayReqT.expired** /ErrorCount >= MAX_COUNT_ERROR => SequenceID++ ErrorCount++ RESET_LINE_DELAY DelayReqT.start (DELAY_REQ_INTERVAL) DelayReq_Req (Port, SequenceID)	W_CNF
20	W_RSP	**DelayReq_Cnf (D_Port, TStamp, Status)** => ignore	W_RSP
21	W_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port <> Port => ignore	W_RSP
22	W_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && !CHECK_DELAY_RSP_VALID => ignore	W_RSP
23	W_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && CHECK_DELAY_RSP_VALID && CHECK_DELAY_RSP_WITH_FU_RSP => Delay := PTCP_Delay Rx_TStamp_T4 := TStamp	W_FU_RSP
24	W_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && CHECK_DELAY_RSP_VALID && !CHECK_DELAY_RSP_WITH_FU_RSP => Delay := PTCP_Delay Rx_TStamp_T4 := TStamp CALC_LINE_DELAY ErrorCount := 0	W_TIMER
25	W_RSP	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	W_RSP

#	Current State	Event /Condition =>Action	Next State
26	W_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID <> Port => ignore	W_CNF
27	W_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type <> FULL_DUPLEX => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
28	W_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && LINK_Status == DOWN => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
29	W_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type == Port_MAU_Type && LINK_Status == Up => ignore	W_RSP
30	W_RSP	**DelayReqT.expired** /ErrorCount < MAX_COUNT_ERROR => SequenceID++ ErrCount++ DelayReqT.start (DELAY_REQ_INTERVAL) DelayReq_Req (Port, SequenceID)	W_CNF
31	W_RSP	**DelayReqT.expired** /ErrorCount >= MAX_COUNT_ERROR => SequenceID++ ErrCount++ RESET_LINE_DELAY DelayReqT.start (DELAY_REQ_INTERVAL) DelayReq_Req (Port, SequenceID)	W_CNF
32	W_FU_RSP	**DelayReq_Cnf (D_Port, TStamp, Status)** => ignore	W_FU_RSP
33	W_FU_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port <> Port => ignore	W_FU_RSP
34	W_FU_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && !CHECK_DELAY_RSP_VALID => ignore	W_FU_RSP
35	W_FU_RSP	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && CHECK_DELAY_RSP_VALID => RESET_LINE_DELAY	W_TIMER
36	W_FU_RSP	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port <> Port => ignore	W_FU_RSP
37	W_FU_RSP	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && !CHECK_DELAY_FU_RSP_VALID => ignore	W_FU_RSP

#	Current State	Event /Condition =>Action	Next State
38	W_FU_RSP	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && CHECK_DELAY_FU_RSP_VALID => Delay := Delay + PTCP_Delay CALC_LINE_DELAY ErrorCount := 0	W_TIMER
39	W_FU_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID <> Port => ignore	W_FU_RSP
40	W_FU_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type <> FULL_DUPLEX => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
41	W_FU_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && LINK_Status == DOWN => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
42	W_FU_RSP	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up => ignore	W_FU_RSP
43	W_FU_RSP	**DelayReqT.expired** /ErrorCount < MAX_COUNT_ERROR => SequenceID++ DelayReqT.start (DELAY_REQ_INTERVAL) ErrorCount++ DelayReq_Req (Port, SequenceID)	W_CNF
44	W_FU_RSP	**DelayReqT.expired** /ErrorCount >= MAX_COUNT_ERROR => SequenceID++ ErrorCount++ RESET_LINE_DELAY DelayReqT.start (DELAY_REQ_INTERVAL) DelayReq_Req (Port, SequenceID)	W_CNF
45	W_TIMER	**DelayReqT.expired** => SequenceID++ DelayReqT.start (DELAY_REQ_INTERVAL) DelayReq_Req (Port, SequenceID)	W_CNF
46	W_TIMER	**DelayReq_Cnf (D_Port, TStamp, Status)** => ignore	W_TIMER
47	W_TIMER	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port <> Port => ignore	W_TIMER
48	W_TIMER	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && !CHECK_DELAY_RSP_VALID => ignore	W_TIMER

#	Current State	Event /Condition =>Action	Next State
49	W_TIMER	**DelayResp_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port && CHECK_DELAY_RSP_VALID => RESET_LINE_DELAY	W_TIMER
50	W_TIMER	**DelayFuResp_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	W_TIMER
51	W_TIMER	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID <> Port => ignore	W_TIMER
52	W_TIMER	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type <> FULL_DUPLEX => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
53	W_TIMER	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && LINKStatus == DOWN => DelayReqT.stop SequenceID++ RESET_LINE_DELAY	DOWN
54	W_TIMER	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status)** /PortID == Port && MAU_Type == Port_MAU_Type && LINK_Status == Up => ignore	W_TIMER

4.4.4.2.1.4 Macros

The Table 68 contains the macros used by DelayRequest.

Table 68 – Macros used by DelayRequest

Name	Meaning
CHECK_DELAY_RSP_VALID	Check parameter of PTCP_DelayResPDU PTCP_Sequence == SequneceID PTCP_RequestSourceAddress == Local_SA PTCP_RequestPortID == Port
CHECK_DELAY_RSP_WITH_FU_RSP	Check FrameID == 0xFF41of PTCP_DelayResPDU
CHECK_DELAY_RSP_HIGH_ACCURACY	Check parameter for high accuracy of PTCP_DelayResPDU PTCP_SubdomainUUID := Subdomain UUID of PTCP_DelayReqPDU PTCP_MasterSourceAddress == Sync master source MAC of PTCP_DelayReqPDU PTCP_SyncID == SyncID of PTCP_DelayReqPDU
CALC_LINE_DELAY	Line delay is the arithmetical average value of the last 8 delay measurements
RESET_LINE_DELAY	Set line delay to zero

4.4.4.2.1.5 Functions

The Table 69 contains the functions used by DelayRequest.

Table 69 – Functions used by DelayRequest

Name	Meaning
DelayReq_Req (Port, SequenceID)	Create PTCP-PDU according PTCP_DelayReqPDU Assignments: D_Port := Port DA :=PTCP_MulticastMACadd for PTCP_DelayReqPDU SA := local source address PTCP_SequenceID := SequenceID PTCP_SubdomainUUID := Subdomain UUID of smallest used SyncID (defalut: 0x0) PTCP_MasterSourceAddress:= Sync master source MAC address of smallest supported SyncID (defalut: 0x0) PTCP_SyncID := Smallest used SyncID (default: 0x0) PTCP_RequestPortID := Port A_SDU := LT, FrameID, PTCP_DelayReqPDU LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VALN_Prio, VLANID, A_SDU)
DelayReq_Cnf (D_Port, TStamp, Status)	LMPM_A_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
DelayResp_Ind (S_Port, TStamp, DelayResPDU)	Receive PTCP-PDU according PTCP_DelayResPDU LMPM_A_Data.ind (CREP, S_Port, TSamp, DA, SA, VLANPrio, VLANID, A_SDU) Assignments: DelayResPDU := A_SDU without LT and FrameID
DelayFuResp_Ind (S_Port, TStamp, DelayFuResPDU)	Receive PTCP-PDU according PTCP_DelayFuResPDU LMPM_A_Data.ind (CREP, D_Port, TSamp, DA, SA, VLANPrio, VLANID, A_SDU) Assignments: DelayFuResPDU := A_SDU without LT and FrameID

4.4.4.2.1.5.1 DelayReq_Req, DelayReq_Cnf, DelayResp_Ind and DelayFuResp_Ind

Local macros used to send PTCP_DelayReqPDU or to receive PTCP_DelayResPDU or PTCP_DelayFuResPDU.

4.4.4.2.2 Line Delay Response Protocol Machine (DelayResponse)

4.4.4.2.2.1 State machine description

Each delay request message shall be immediately respondet with a delay response messages. The state machine which receives a delay request frame is called DelayResponder.

The following Figure 30 shows the delay response state machine:

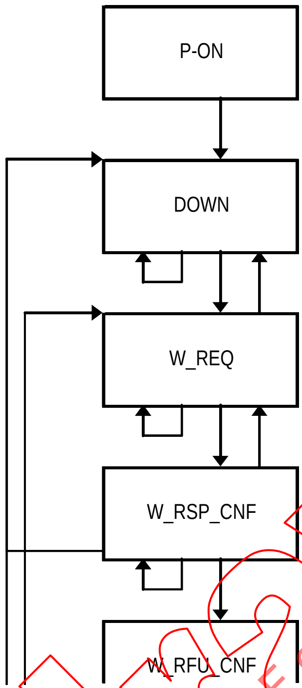

Figure 30 – State transition diagram of delay response

States of the DelayResponse

POWER_ON
Data initialization.

DOWN
Link of port is down. Wait for link up.

W_REQ
If a delay request message is received the receive time stamp will be stored an the DelayResonder sends a delay response message.

W_RSP_CNF
Wait for the confirmation of the delay response message, calculate the delay response time and send a delay response follow up message.

W_RFU_CNF
Wait for the confirmation of the delay response follow up message.

Local variables of the DelayResponse

Rx_TStamp_T2
This local variable contains the recieve time stamp of the delay request message.

Tx_TStamp_T3
This local variable contains the transmit time stamp of the delay response message.

4.4.4.2.2.2 DelayResponse state table

Table 70 contains the state table used by DelayResponse.

Table 70 – DelayResponse state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> Rx_TStamp_T2 := NIL Tx_TStamp_T3 := NIL	DOWN
2	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID <> Port => ignore	DOWN
3	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID <> Port && MAU_Type <> FULL_DUPLEX => ignore	DOWN
4	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status <> Up => ignore	DOWN
5	DOWN	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up =>	W_REQ
6	DOWN	**DelayReq_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	DOWN
7	DOWN	**DelayResp_Cnf (D_Port, TStamp, Status)** => ignore	DOWN
8	DOWN	**DelayFuResp_Cnf (D_Port, TStamp, Status)** => ignore	DOWN
9	W_REQ	**DelayReq_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port <> Port => ignore	W_REQ

#	Current State	Event /Condition =>Action	Next State
10	W_REQ	**DelayReq_Ind (S_Port, TStamp, PTCP_PDU)** /S_Port == Port => store PTCP_PDU in DReq_PDU Rx_TStamp_T2 := TStamp DelayResp_Req (Port, DReq_PDU)	W_RSP_CNF
11	W_REQ	**DelayResp_Cnf (D_Port, TStamp, Status)** => ignore	W_REQ
12	W_REQ	**DelayFuResp_Cnf (D_Port, TStamp, Status)** => ignore	W_REQ
13	W_REQ	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID <> Port => ignore	W_REQ
14	W_REQ	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type <> FULL_DUPLEX =>	DOWN
15	W_REQ	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && LINK_Status == DOWN =>	DOWN
16	W_REQ	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up => ignore	W_REQ
17	W_RSP_CNF	**DelayReq_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	W_RSP_CNF
18	W_RSP_CNF	**DelayResp_Cnf (D_Port, TStamp, Status)** /Status <> OK =>	W_REQ
19	W_RSP_CNF	**DelayResp_Cnf (D_Port, TStamp, Status)** /Status == OK => Tx_TStamp_T3 := TStamp ResTime := CALC_RESIDENTIAL_TIME DelayFuResp_Req (Port, ResTime, DReq_PDU)	W_RFU_CNF

#	Current State	Event /Condition =>Action	Next State
20	W_RSP_CNF	**DelayFuResp_Cnf (D_Port, TStamp, Status)** => ignore	W_RSP_CNF
21	W_RSP_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID <> Port => ignore	W_RSP_CNF
22	W_RSP_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type <> FULL_DUPLEX =>	DOWN
23	W_RSP_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && LINK_Status == DOWN =>	DOWN
24	W_RSP_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up => ignore	W_RSP_CNF
25	W_RFU_CNF	**DelayReq_Ind (S_Port, TStamp, PTCP_PDU)** => ignore	W_RFU_CNF
26	W_RFU_CNF	**DelayResp_Cnf (D_Port, TStamp, Status)** => ignore	W_RFU_CNF
27	W_RFU_CNF	**DelayFuResp_Cnf (D_Port, TStamp, Status)** =>	W_REQ
28	W_RFU_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID <> Port => ignore	W_RFU_CNF
29	W_RFU_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type <> FULL_DUPLEX =>	DOWN

#	Current State	Event /Condition =>Action	Next State
30	W_RFU_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && LINK_Status == DOWN =>	DOWN
31	W_RFU_CNF	**MAUType_Change.ind (PortID, MAU_Type, LINK_Status** /PortID == Port && MAU_Type == FULL_DUPLEX && LINK_Status == Up => ignore	W_RFU_CNF

4.4.4.2.2.3 Macros

The Table 71 contains the macros used by DelayResponse.

Table 71 – Macros used by DelayResponse

Name	Meaning
CALC_RESIDENTIAL_TIME	Calculate the residential time ResTime := (Tx_Stamp_T3 - Rx_Stamp_T2) * RateCompensationFactor

4.4.4.2.2.4 Functions

The Table 72 contains the functions used by the DelayResponse.

Table 72 – Functions used by DelayResponse

Name	Meaning
DelayResp_Req (Port, PTCP_DelayReqPDU)	Create PTCP-PDU according PTCP_DelayResPDU Assignments: D_Port := Port DA :=PTCP_MulticastMACadd for PTCP_DelayResPDU SA := local source address PTCP_SequenceID := SequenceID of PTCP_DelayReqPDU PTCP_SubdomainUUID := if supported Subdomain UUID of PTCP_DelayReqPDU else Subdomain of smallest supported SyncID or defalut: 0x0 PTCP_MasterSourceAddress:= if SyncID of PTCP_DelayReqPDU is supported source MAC address of sync master else source MAC address of sync master with smallest SyncID or defalut: 0x0 PTCP_SyncID := if supported SyncID of PTCP_DelayReqPDU else SyncID of smallest supported SyncID or defalut: 0x0 PTCP_RequestPortID := PortID of PTCP_DelayResPDU A_SDU := LT, FrameID, PTCP_DelayResPDU LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VALN_Prio, VLANID, A_SDU)
DelayFuResp_Req (Port, ResTime, DReq_PDU)	Create PTCP-PDU according PTCP_DelayFuResPDU Assignments: D_Port := Port DA := PTCP_MulticastMACadd for PTCP_DelayFuResPDU SA := local source address PTCP_Delay := ResTime A_SDU := LT, FrameID, PTCP_DelayFuResPDU LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VALN_Prio, VLANID, A_SDU)
DelayResp_Cnf (D_Port, TStamp, Status)	LMPM_A_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
DelayFuResp_Cnf (D_Port, TStamp, Status)	LMPM_A_Data.conf (CREP, D_Port, TStamp, LMPM_Status) Assignments: Status := LMPM_status
DelayReq_Ind (S_Port, TStamp, DelayReqPDU)	Receive PTCP-PDU according PTCP_DelayReqPDU LMPM_A_Data.ind (CREP, S_Port, TSamp, DA, SA, VLANPrio, VLANID, A_SDU) Assignments: DelayReqPDU := A_SDU without LT and FrameID

4.4.4.2.2.4.1 DelayResp_Req, DelayFuResp_Req, DelayResp_Cnf, DelayFuResp and DelayReq_Ind

Local macros used to send PTCP_DelayResPDU and PTCP_DelayFuResPDU or to receive PTCP_DelayReqPDU.

4.4.4.3 Overview for master slave state tables

Figure 31 shows an overview of the interaction between the PTCP state tables.

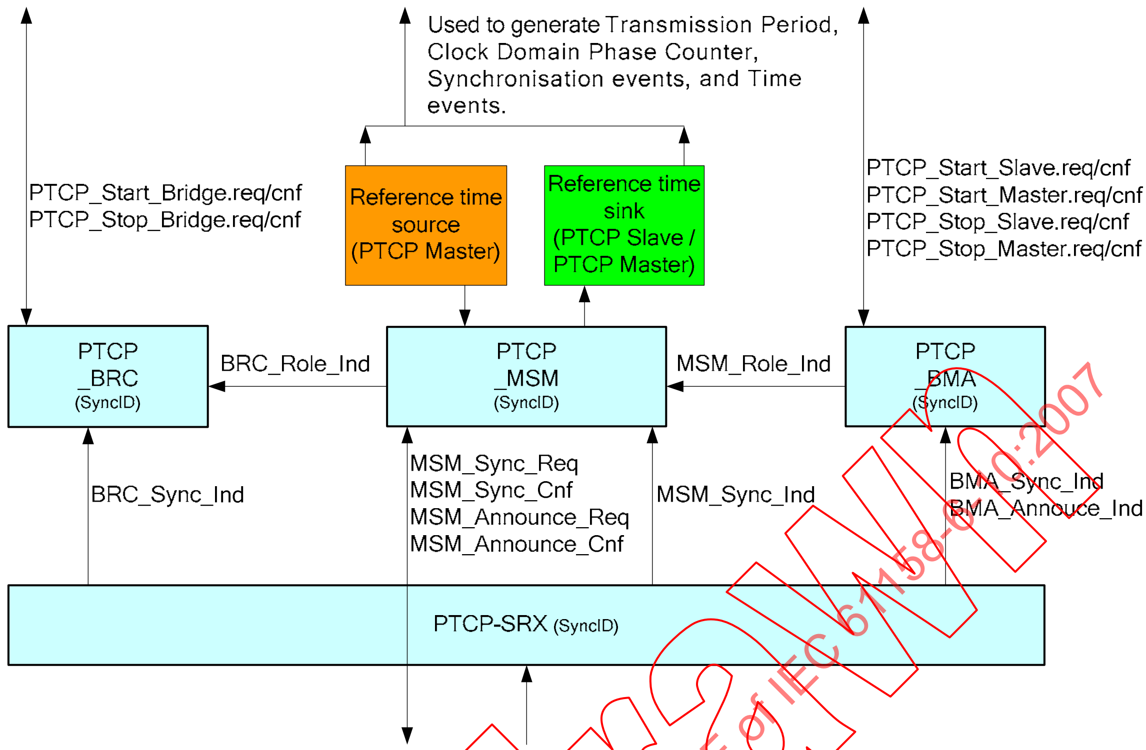

Figure 31 – Overview of PTCP

4.4.4.4 Best-Master-Algorithm Protocol Machine (BMA)

4.4.4.4.1 Primitive definitions

4.4.4.4.1.1 Primitives exchanged between BMA and ASE

The service primitives including their associated parameters issued by BMA user received by BMA and vice versa are described in the PTCP ASE in the service definition.

4.4.4.4.2 State machine description

The BMA-state-machine shall elect the master and secondary master from all announced master-capable devices. These decisions are made local by every device. An instance of the state machine exists for every PTCP SyncID.

Due to the fact, that not every device is able to be master or should be master, the behaviour of the state machine depends on his role. The role is set by PTCP services. Roles are master, secondary master, slave and bridge. A master also needs slave functionality and shall be first started as slave (PTCP_Start_Slave.req) and after this as master (PTCP_Start_Master.req). The stop sequence shall be vice versa.

Whether a device with the role master become master or secondary master will be decided by the best master algorithm. Every potential master has to send announce messages to take part in master and secondary master election. A device sends announce messages, if it has no master or secondary master elected.

The master election will be made in two steps. With the first received annaunce or synchronization message the potentially master will be stored in a remote list. With the next annaunce or synchronization message the BMA elects this master, if it's the best known master.

During master competition all potentially masters will be collected in the remote list. After election they drop out via altering.

If the decision to be master is dropped, the MSM shall make the transfer from slave to master. An active master starts to send sync messages and stops to send announce messages. A secondary master proceeds sending announce messages. The remaining potential masters stop to send announce messages and have to be slaves. If the secondary master was removed from the sub domain or take over the master role (master was removed), the potential masters restart sending announce messages.

The following Figure 32 shows the BMA state machine:

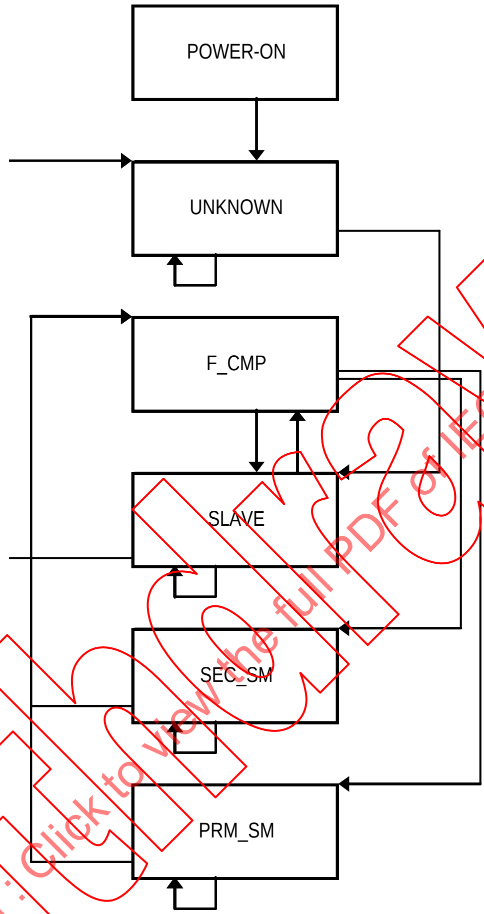

Figure 32 – State transition diagram of BMA

States of the BMA

POWER_ON
Data initialization

UNKNOWN
The synchronization role is unknown. A role shall set via PTCP services.

SLAVE
The device shall synchronize using the synchronization messages of the master.

SEC_SM
At this state the device is, due to its synchronization attributes, elected as the second best master. The secondary master sends announce messages.

PRM_SM
At this state the device is, due to its synchronization attributes, elected as the best master. The master shall send synchronization messages.

F_CMP
This is a transition state between the states SLAVE, SEC_SM and PRM_SM. The elected primary master source address and secondary master source address will be compared with the own source address.

Local variables of the BMA

LocSlaveData (structue)

Subdomain (UUID)
This local variable contains the unique identifier of the PTCP subdomain.

SendInterval (Unsigned16)
This local variable contains the send interval for sync messages.

LocMasterData (structure)

Class (Unsigned8)
This local variable contains the value of the synchronization class.

Stratum (Unsigned8)
This local variable contains the quality of the time provided by this device.

Variance (Integer16)
This local variable characterizes the quality of a clock provided by this device.

SA (OctetString[6])
This local variable contains the MAC address of this device.

SendInterval (Unsigned16)
This local variable contains the send interval for sync messages.

RPrmM_List{SM_SA} (structure)

Class (Unsigned8)
This local variable contains the value of the synchronization class.

Stratum (Unsigned8)
This local variable contains the quality of the time provided by this master.

Variance (Integer16)
This local variable characterizes the quality of a clock provided by this master.

SA (OctetString[6])
This local variable contains the MAC address of this master.

Receipt (Unsigned16)
Receive counter for synchronization or announce messages from this master. This counter shall be used for aging.

RSecM_List{SM_SA} (structure)

Class (Unsigned8)
This local variable contains the value of the synchronization class.

Stratum (Unsigned8)
This local variable contains the quality of the time provided by this master.

Variance (Integer16)
This local variable characterizes the quality of a clock provided by this master.

SA (OctetString[6])
This local variable contains the MAC address of this master.

Receipt (Unsigned16)
Receive counter for synchronization or announce messages from this master. This counter shall be used for aging.

PrmM (OctetString[6])
This local variable contains the MAC address of the primary master.

SecM (OctetString[6])
This local variable contains the MAC address of the secondary master.

4.4.4.4.3 BMA state table

Table 73, Table 74, and Table 75 contains the state table used by BMA.

Table 73 – BMA state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> ini LocSlaveData ini LocMasterData ini PrmM ini SecM ini RPrmM_List ini RSecM_List	UNKNOWN
2	UNKNOWN	**ReceiptTimer.expired (SyncID)** => ignore	UNKNOWN
3	UNKNOWN	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** => ignore	UNKNOWN
4	UNKNOWN	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** => ignore	UNKNOWN
5	UNKNOWN	**PTCP_Start_Slave.req (SyncID, Subdomain, SendInterval)** => INI_LOCAL_SLAVE_DATA_SET ReceiptTimer.start (SyncID, BMA_RCV_TIME_OUT) ErrCode := OK PTCP_Start_Slave.cnf (SyncID, ErrCode) MSM_Role_Ind (SyncID, SLAVE)	SLAVE
6	UNKNOWN	**PTCP_Stop_Slave.req (SyncID)** => ErrCode := SLAVE_NOT_EXISTS PTCP_Stop_Slave.cnf (SyncID, ErrCode)	UNKNOWN
7	UNKNOWN	**PTCP_Start_Master.req (SyncID, Subdomain, Class, Stratum, Variance, SendInterval)** => ErrCode := SLAVE_NOT_EXISTS PTCP_Start_Master.cnf (SyncID, ErrCode)	UNKNOWN
8	UNKNOWN	**PTCP_Stop_Master.req (SyncID)** => ErrCode := MASTER_NOT_EXISTS PTCP_Stop_Master.cnf (SyncID, ErrCode)	UNKNOWN
9	F_CMP	/LocMasterData.SA == NIL => MSM_Role_Ind (SyncID, PrmM.SA, SLAVE)	SLAVE
10	F_CMP	/LocMasterData.SA <> NIL && PrmM.SA == LocMasterData.SA => MSM_Role_Ind (SyncID, PrmM.SA, PRM_SM)	PRM_SM
11	F_CMP	/LocMasterData.SA <> NIL&& PrmM.SA <> LocMasterData.SA&& SecM.SA == LocMasterData.SA=>MSM_Role_Ind (SyncID, PrmM.SA, SEC_SM)	SEC_SM
12	F_CMP	/LocMasterData.SA <> NIL && PrmM.SA <> LocMasterData.SA && SecM.SA <> LocMasterData.SA => MSM_Role_Ind (SyncID, PrmM.SA, SLAVE)	SLAVE

#	Current State	Event /Condition =>Action	Next State
13	SLAVE	**ReceiptTimer.expired (SyncID)** => UPDATE_PRM_SEC_SM ReceiptTimer.start (SyncID, BMA_RCV_TIME_OUT)	F_CMP
14	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /!CHECK_SUBDOMAIN_SUPPORTED => ignore	SLAVE
15	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA == Local_SA => ignore	SLAVE
16	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_FROM_SEC_SM => PUT_IN_RPRM_SM_LIST REM_SEC_SM UPDATE_SEC_SM	F_CMP
17	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_FROM_SEC_SM && !CHECK_IN_RPRM_SM_LIST && !CHECK_IN_RSEC_SM_LIST => PUT_IN_RPRM_SM_LIST	SLAVE
18	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)**/CHECK_SUBDOMAIN_SUPPORTED&& SM_SA <> Local_SA&& !CHECK_FROM_SEC_SM&& CHECK_IN_RPRM_SM_LIST&& !CHECK_IN_RSEC_SM_LIST&& CHECK_FROM_PRM_SM=>UPDATE_IN_RPRM_SM_LIST	SLAVE
19	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_FROM_SEC_SM && CHECK_IN_RPRM_SM_LIST && !CHECK_IN_RSEC_SM_LIST && !CHECK_FROM_PRM_SM && !CHECK_BEST_IS_NEW_PRM_SM => UPDATE_IN_RPRM_SM_LIST	SLAVE
20	SLAVE	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_FROM_SEC_SM && CHECK_IN_RPRM_SM_LIST && !CHECK_IN_RSEC_SM_LIST && !CHECK_FROM_PRM_SM && CHECK_BEST_IS_NEW_PRM_SM => UPDATE_IN_RPRM_SM_LIST UPDATE_PRM_SM_FROM_RPRM	SLAVE

#	Current State	Event /Condition =>Action	Next State
21	SLAVE	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /!CHECK_SUBDOMAIN_SUPPORTED => ignore	SLAVE
22	SLAVE	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA == Local_SA => ignore	SLAVE
23	SLAVE	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RPRM_SM_LIST && !CHECK_IN_RSEC_SM_LIST => PUT_IN_RSEC_SM_LIST	SLAVE
24	SLAVE	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RPRM_SM_LIST && CHECK_IN_RSEC_SM_LIST && CHECK_FROM_SEC_SM => UPDATE_IN_RSEC_SM_LIST	SLAVE
25	SLAVE	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RPRM_SM_LIST && CHECK_IN_RSEC_SM_LIST && !CHECK_FROM_SEC_SM && !CHECK_BEST_IS_NEW_SEC_SM => UPDATE_IN_RSEC_SM_LIST	SLAVE
26	SLAVE	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RPRM_SM_LIST && CHECK_IN_RSEC_SM_LIST && !CHECK_FROM_SEC_SM && CHECK_BEST_IS_NEW_SEC_SM => UPDATE_IN_RSEC_SM_LIST UPDATE_SEC_SM_FROM_RSEC	SLAVE
27	SLAVE	**PTCP_Start_Master.req (SyncID, Subdomain, Class, Stratum, Variance, SendInterval)** /CHECK_LOCAL_MASTER_DATA_EXISTS => ErrCode := MASTER_NOT_POSSIBLE PTCP_Start_Master.cnf (SyncID, ErrCode)	SLAVE
28	SLAVE	**PTCP_Start_Master.req (SyncID, Subdomain, Class, Stratum, Variance, SendInterval)** /!CHECK_LOCAL_MASTER_DATA_EXISTS => INI_LOCAL_MASTER_DATA_SET ErrCode := OK PTCP_Start_Master.cnf (SyncID, ErrCode)	SLAVE

#	Current State	Event /Condition =>Action	Next State
29	SLAVE	**PTCP_Stop_Master.req (SyncID)** /!CHECK_LOCAL_MASTER_DATA_EXISTS => ErrCode := MASTER_NOT_EXISTS PTCP_Start_Master.cnf (SyncID, ErrCode)	SLAVE
30	SLAVE	**PTCP_Stop_Master.req (SyncID)** /CHECK_LOCAL_MASTER_DATA_EXISTS => REM_LOCAL_MASTER_DATA_SET ErrCode := OK PTCP_Start_Master.cnf (SyncID, ErrCode)	SLAVE
31	SLAVE	**PTCP_Start_Slave.req (SyncID, Subdomain, SendInterval)** => ErrCode := SLAVE_EXISTS PTCP_Start_Slave.cnf (SyncID, ErrCode)	SLAVE
32	SLAVE	**PTCP_Stop_Slave.req (SyncIDl)** => REM_LOCAL_SLAVE_DATA_SET ErrCode := OK PTCP_Stop_Slave.cnf (SyncID, ErrCode)	UNKNOWN
33	SEC_SM	**ReceiptTimer.expired (SyncID)** => UPDATE_PRM_SEC_SM ReceiptTimer.start (SyncID, BMA_RCV_TIME_OUT)	F_CMP
34	SEC_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /!CHECK_SUBDOMAIN_SUPPORTED => ignore	SEC_SM
35	SEC_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA == Local_SA => ignore	SEC_SM
36	SEC_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RPRM_SM_LIST => PUT_IN_RPRM_SM_LIST	SEC_SM
37	SEC_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RPRM_SM_LIST && CHECK_FROM_PRM_SM => UPDATE_IN_RPRM_SM_LIST	SEC_SM
38	SEC_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RPRM_SN_LIST && !CHECK_FROM_PRM_SM && !CHECK_BEST_IS_NEW_PRM_SM => UPDATE_IN_RPRM_SM_LIST	SEC_SM

#	Current State	Event /Condition =>Action	Next State
39	SEC_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RPRM_SN_LIST && !CHECK_FROM_PRM_SM && CHECK_BEST_IS_NEW_PRM_SM => UPDATE_IN_RPRM_SM_LIST UPDATE_PRM_SM_FROM_RPRM	SEC_SM
40	SEC_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /!CHECK_SUBDOMAIN_SUPPORTED => ignore	SEC_SM
41	SEC_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA == Local_SA => ignore	SEC_SM
42	SEC_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RSEC_SM_LIST => PUT_IN_RSEC_SM_LIST	SEC_SM
43	SEC_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RSEC_SM_LIST && !CHECK_BEST_IS_NEW_SEC_SM => UPDATE_IN_RSEC_SM_LIST	SEC_SM
44	SEC_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RSEC_SM_LIST && CHECK_BEST_IS_NEW_SEC_SM => UPDATE_IN_RSEC_SM_LIST UPDATE_SEC_SM	F_CMP
45	SEC_SM	**PTCP_Start_Master.req (SyncID, Subdomain, Class, Stratum, Variance, SendInterval)** => ErrCode := MASTER_EXISTS PTCP_Start_Master.cnf (SyncID, ErrCode)	SEC_SM
46	SEC_SM	**PTCP_Stop_Master.req (SyncID)** => REM_LOCAL_MASTER_DATA_SET UPDATE_SEC_SM ErrCode := OK PTCP_Stop_Master.cnf (SyncID, ErrCode)	F_CMP
47	SEC_SM	**PTCP_Start_Slave.req (SyncID, Subdomain, SendInterval)** => ErrCode := SLAVE_EXISTS PTCP_Start_Slave.cnf (SyncID, ErrCode)	SEC_SM
48	SEC_SM	**PTCP_Stop_Slave.req (SyncIDl)** => ErrCode := WRONG_SEQUENCE PTCP_Stop_Slave.cnf (SyncID, ErrCode)	SEC_SM

#	Current State	Event /Condition =>Action	Next State
49	PRM_SM	**ReceiptTimer.expired (SyncID)** => UPDATE_REMOTE_SM_LIST ReceiptTimer.start (SyncID, BMA_RCV_TIME_OUT)	F_CMP
50	PRM_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /!CHECK_SUBDOMAIN_SUPPORTED => ignore	PRM_SM
51	PRM_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA == Local_SA => ignore	PRM_SM
52	PRM_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RPRM_SM_LIST => PUT_IN_RPRM_SM_LIST	PRM_SM
53	PRM_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RPRM_SM_LIST && CHECK_FROM_PRM_SM => UPDATE_IN_RPRM_SM_LIST	PRM_SM
54	PRM_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RPRM_SN_LIST && !CHECK_BEST_IS_NEW_PRM_SM => UPDATE_IN_RPRM_SM_LIST	PRM_SM
55	PRM_SM	**BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RPRM_SN_LIST && CHECK_BEST_IS_NEW_PRM_SM => UPDATE_IN_RPRM_SM_LIST UPDATE_PRM_SEC_SM	F_CMP
56	PRM_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /!CHECK_SUBDOMAIN_SUPPORTED => ignore	PRM_SM
57	PRM_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA == Local_SA => ignore	PRM_SM

#	Current State	Event /Condition =>Action	Next State
58	PRM_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && !CHECK_IN_RSEC_SM_LIST => PUT_IN_RSEC_SM_LIST	PRM_SM
59	PRM_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RSEC_LIST && CHECK_FROM_SEC_SM => UPDATE_IN_RSEC_SM_LIST	PRM_SM
60	PRM_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RSEC_LIST && !CHECK_FROM_SEC_SM && !CHECK_BEST_IS_NEW_SEC_SM => UPDATE_IN_RSEC_SM_LIST	PRM_SM
61	PRM_SM	**BMA_Announce_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance)** /CHECK_SUBDOMAIN_SUPPORTED && SM_SA <> Local_SA && CHECK_IN_RSEC_LIST && !CHECK_FROM_SEC_SM && CHECK_BEST_IS_NEW_SEC_SM => UPDATE_IN_RSEC_SM_LIST UPDATE_SEC_SM_FROM_RSEC	PRM_SM
62	PRM_SM	**PTCP_Start_Master.req (SyncID, Subdomain, Class, Stratum, Variance, SendInterval)** => ErrCode := MASTER_EXISTS PTCP_Start_Master.cnf (SyncID, ErrCode)	PRM_SM
63	PRM_SM	**PTCP_Stop_Master.req (SyncID)** => REM_LOCAL_MASTER_DATA_SET UPDATE_PRM_SM ErrCode := OK PTCP_Stop_Master.cnf (SyncID, ErrCode)	F_CMP
64	PRM_SM	**PTCP_Start_Slave.req (SyncID, Subdomain, SendInterval)** => ErrCode := SLAVE_EXISTS PTCP_Start_Slave.cnf (SyncID, ErrCode)	PRM_SM
65	PRM_SM	**PTCP_Stop_Slave.req (SyncIDl)** => ErrCode := WRONG_SEQUENCE PTCP_Stop_Slave.cnf (SyncID, ErrCode)	PRM_SM

Table 74 – BMA state table – check remote

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**GetBestRSM (SyncID, RmoM_List)** /RmoM_List.Entry{SyncID}.Num_entry == 0 => Best_RSM_SA := NIL	IDLE
2	IDLE	**GetBestRSM (SyncID, RmoM_List)** /RmoM_List.Entry{SyncID}.Num_entry == 1 => N_BestM := 0 Best_RSM_SA := RmoM_List.Entry{SyncID, N_BestM}.SA Result := OK	IDLE
3	IDLE	**GetBestRSM (SyncID, RmoM_List)** /RmoM_List.Entry{SyncID}.Num_entry > 1 => N_BestM := 0 N_CmpM := 1	CMP_R_C
4	CMP_R_C	/N_CmpM < RmoM_List.Entry{SyncID}.Num_entry && RmoM_List.Entry{SyncID}[N_BestM].Class < RmoM_List.Entry{SyncID}[N_CmpM].Class => N_CmpM++	CMP_R_C
5	CMP_R_C	/N_CmpM < RmoM_List.Entry{SyncID}.Num_entry && RmoM_List.Entry{SyncID}[N_BestM].Class > RmoM_List.Entry{SyncID}[N_CmpM].Class => N_BestM := N_CmpM N_CmpM++	CMP_R_C
6	CMP_R_C	/N_CmpM < RmoM_List.Entry{SyncID}.Num_entry && RmoM_List.Entry{SyncID}[N_BestM].Class == RmoM_List.Entry{SyncID}[N_CmpM].Class =>	CMP_R_S
7	CMP_R_C	/N_CmpM >= RmoM_List.Entry{SyncID}.Num_entry => Best_RSM_SA := RmoM_List.Entry{SyncID, N_BestM }.SA	IDLE
8	CMP_R_S	/RmoM_List.Entry{SyncID}[N_BestM].Stratum < RmoM_List.Entry{SyncID}[N_CmpM].Stratum => N_CmpM++	CMP_R_C
9	CMP_R_S	/RmoM_List.Entry{SyncID}[N_BestM].Stratum > RmoM_List.Entry{SyncID}[N_CmpM].Stratum=>N_BestM := N_CmpMN_CmpM++	CMP_R_C
10	CMP_R_S	/RmoM_List.Entry{SyncID}[N_BestM].Stratum == RmoM_List.Entry{SyncID}[N_CmpM].Stratum =>	CMP_R_V
11	CMP_R_V	/N_CmpM < RmoM_List.Entry{SyncID}.Num_entry && RmoM_List.Entry{SyncID}[N_BestM].Variance < RmoM_List.Entry{SyncID}[N_CmpM].Variance => N_CmpM++	CMP_R_C
12	CMP_R_V	/RmoM_List.Entry{SyncID}[N_BestM].Variance > RmoM_List.Entry{SyncID}[N_CmpM].Variance => N_BestM := N_CmpM N_CmpM++	CMP_R_C

#	Current State	Event /Condition =>Action	Next State
13	CMP_R_V	/RmoM_List.Entry{SyncID}[N_BestM].Variance == RmoM_List.Entry{SyncID}[N_CmpM].Variance =>	CMP_R_SA
14	CMP_R_SA	/RmoM_List.Entry{SyncID}[N_BestM].SA < RmoM_List.Entry{SyncID}[N_CmpM].SA => N_CmpM++	CMP_R_C
15	CMP_R_SA	/RmoM_List.Entry{SyncID}[N_BestM].SA > RmoM_List.Entry{SyncID}[N_CmpM].SA => N_BestM := N_CmpM N_CmpM++	CMP_R_C
16	CMP_R_SA	/RmoM_List.Entry{SyncID}[N_BestM].SA == RmoM_List.Entry{SyncID}[N_CmpM].SA => N_CmpM++	CMP_R_C

Table 75 – BMA state table – check local vs. remote

#	Current State	Event /Condition =>Action	Next State
1	IDLE	**GetBestSM (SM_1, SM_2)** /SM_1 == NIL && SM_2.SA == NIL => BetterSM_SA := NIL	IDLE
2	IDLE	**GetBestSM (SM_1, SM_2)** /SM_1 == NIL && SM_2.SA != NIL => BetterSM_SA := SM_2.SA	IDLE
3	IDLE	**GetBestSM (SM_1, SM_2)** /SM_1 != NIL && SM_2.SA != NIL =>	CMP_R_L_C
4	CMP_R_L_C	/SM_1.Class < SM_2.Class => BetterSM_SA := SM_1.SA	IDLE
5	CMP_R_L_C	/SM_1.Class > SM_2.Class => BetterSM_SA := SM_2.SA	IDLE
6	CMP_R_L_C	/SM_1.Class == SM_2.Class =>	CMP_R_L_S
7	CMP_R_L_S	/SM_1.Stratum < SM_2.Stratum => BetterSM_SA := SM_1.SA	IDLE

#	Current State	Event /Condition =>Action	Next State
8	CMP_R_L_S	/SM_1.Stratum > SM_2.Stratum => BetterSM_SA := SM_2.SA	IDLE
9	CMP_R_L_S	/SM_1.Stratum == SM_2.Stratum =>	CMP_R_L_V
10	CMP_R_L_V	/SM_1.Variance < SM_2.Variance=>BetterSM_SA := SM_1.SA	IDLE
11	CMP_R_L_V	/SM_1.Variance > SM_2.Variance => BetterSM_SA := SM_2.SA	IDLE
12	CMP_R_L_V	/SM_1.Variance == SM_2.Variance =>	CMP_R_L_SA
13	CMP_R_L_SA	/SM_1.SA < SM_2.SA => BetterSM_SA := SM_1.SA	IDLE
14	CMP_R_L_SA	/SM_1.SA > SM_2.SA => BetterSM_SA := SM_2.SA	IDLE
15	CMP_R_L_SA	/SM_1.SA == SM_2.SA => BetterSM_SA := SM_1.SA	IDLE

4.4.4.4.4 Macros

Table 76 contains the macros used by BMA.

Table 76 – Macros used by BMA

Name	Meaning
Ini LocSlaveData	Initialize local slave data set LocSlaveData.Subdomain := NIL
INI_LOCAL_SLAVE_DATA_SET	Initialize local slave data set LocSlaveData.Subdomain := Subdomain LocSlaveData.SendInterval := SendInterval
REM_LOCAL_SLAVE_DATA_SET	Remove slave from slave list LocSlaveData.Valid := False
CHECK_SUBDOMAIN_SUPPORTED	Check sync master subdomain is supported LocSlaveData.Subdomain == Subdomain
Ini LocMasterData	Initialize local master data set LocMasterData.SA := NIL
INI_LOCAL_MASTER_DATA_SET	Insert master in local master list LocMasterData.Startum := Startum LocMasterData.Variance := Variance LocMasterData.Class := Class LocMasterData.SA := SM_SA LocMasterData.SendInterval := SendInterval
CHECK_LOCAL_MASTER_DATA_EXISTS	Check sync master data set for local master exists LocMasterData.SA <> NIL
REM_LOCAL_MASTER_DATA_SET	Remove local master LocMasterData.SA := NIL
Ini RPrmM_List	initialize data set for remote primary sync master RPrmM_List.Num_entry := 0
CHECK_IN_RPRM_SM_LIST	Check sync master source address already in remote master list SM_SA in RPrmM_List

PUT_IN_RPRM_SM_LIST	Insert sync master in remote sync master list
	if (RPrmM_List.Num_entry < MAX_CNT_REMOTE_SM)
	  RPrmM_List.Insert(SM_SA)
	  RPrmM_List.Entry{SM_SA}.Stratum := Stratum
	  RPrmM_List.Entry{SM_SA}.Variance := Variance
	  RPrmM_List.Entry{SM_SA}.Class := Class
	  RPrmM_List.Entry{SM_SA}.SA := SM_SA
	  RPrmM_List.Entry{SM_SA}.Receipt := 1
	  RPrmM_List.Num_entry++

REM_FROM_RPRM_SM_LIST	Remove master from reomot primary master list RPrmM_List..Num_entry-- RPrmM_List.Remove(SM_SA)
UPDATE_IN_RPRM_SM_LIST	Update entry in remote sync master list RPrmM_List.Entry{SM_SA}.Stratum := Stratum RPrmM_List.Entry{SM_SA}.Variance := Variance RPrmM_List.Entry{SM_SA}.Class := Class RPrmM_List.Entry{SM_SA}.Receipt += 1

AGING_OF_RPRM_SM_LIST	Aging of entries in remote sync master list
	for m := 0 to RPrmM_List.Num_entry
	  if (RPrmM_List.Entry[m].Receipt == 0
	    REM_FROM_RPRM_SM_LIST
	  else
	    RPrmM_List.Entry[m].Receipt := 0

Ini RSecM_List	initialize data set for remote primary sync master RSecM_List.Num_entry := 0

Name	Meaning
CHECK_IN_RSEC_SM_LIST	Check sync master source address already in remote master list SM_SA in RSecM_List
PUT_IN_RSEC_SM_LIST	Insert sync master in remote sync master list if (RSecM_List.Num_entry < MAX_CNT_REMOTE_SM) RSecM_List.Insert(SM_SA) RSecM_List.Entry{SM_SA}.Stratum := Stratum RSecM_List.Entry{SM_SA}.Variance := Variance RSecM_List.Entry{SM_SA}.Class := Class RSecM_List.Entry{SM_SA}.SA := SM_SA RSecM_List.Entry{SM_SA}.Receipt := 1 RSecM_List.Num_entry++
REM_FROM_RSEC_SM_LIST	Remove master from remote secondary master list RSecM_List.Num_entry-- RSecM_List.Remove(SM_SA)
UPDATE_IN_RSEC_SM_LIST	Update entry in remote sync master list RSecM_List.Entry{SM_SA}.Stratum := Stratum RSecM_List.Entry{SM_SA}.Variance := Variance RSecM_List.Entry{SM_SA}.Class := Class RSecM_List.Entry{SM_SA}.Receipt += 1

AGING_OF_RSEC_SM_LIST	Aging of entries in remote sync master list
	for m := 0 to RSecM_List.Num_entry
	if (RSecM_List.Entry[m].Receipt == 0
	REM_FROM_RSEC_SM_LIST
	else
	RSecM_List.Entry[m].Receipt := 0

Ini PrmM	initialize primary sync master PrmM.SA := NIL
CHECK_FROM_PRM_SM	Check sync master is primary sync master SM_SA == PrmM.SA
UPDATE_PRM_SM_FROM_RPRM	Update primary remote master BestRPrmSM_SA := GetBestRSM (RPrmM_List) PrmM.SA := BestPrmSM_SA
UPDATE_PRM_SM	Update primary sync master BestRPrmSM_SA := GetBestRSM (RPrmM_List) BestPrmSM_SA := GetBestSM (LocMasterData, RPrmM_List.Entry{BestRPrmSM_SA}) PrmM.SA := BestPrmSM_SA
REM_PRM_SM	Remove primary sync master PrmM.SA := NIL
ini SecM	initialize secondary sync master SecM.SA := NIL
CHECK_FROM_SEC_SM	Check sync master is secondary sync master SM_SA == SecM.SA
UPDATE_SEC_SM_FROM_RSEC	Update secondary remote sync master BestRSecSM_SA := GetBestRSM (RSecM_List) SecSM.SA := BestSecSM_SA
UPDATE_SEC_SM	Update secondary sync master BestRSecSM_SA := GetBestRSM (RSecM_List) BestSecSM_SA := GetBestSM (LocMasterData, RPrmM_List.Entry{BestRSecSM_SA}) SecSM.SA := BestSecSM_SA
REM_SEC_SM	Remove secondary sync master from secondary sync master list SecM.SA := NIL

Name	Meaning

CHECK_BEST_IS_NEW_PRM_SM	Compare actuel primary master with new primary master
	if (PrmM.SA == LocMasterData.SA)
	BestPrmSM_SA := GetBestSM (LocMasterData, RPrmM_List.Entry{SM_SA})
	else
	BestPrmSM_SA := GetBestSM (RPrmM_List.Entry{PrmSM.SA}, RPrmM_List.Entry{SM_SA})
	if (PrmSM.SA <> BestPrmSM_SA) TRUE else FALSE

CHECK_BEST_IS_NEW_SEC_SM	Compare actual secondary master with new secondary master
	if (SecM.SA == LocMasterData.SA)
	BestPrmSM_SA := GetBestSM (LocMasterData, RSecM_List.Entry{SM_SA})
	else
	BestPrmSM_SA := GetBestSM (RSecM_List.Entry{PrmSM.SA}, RSecM_List.Entry{SM_SA})
	if (SecSM.SA <> BestSecSM_SA) TRUE else FALSE

UPDATE_PRM_SEC_SM	Aging of remote sync master lists and get best primary and secondary sync master
	AGING_OF_RPRM_SM_LIST
	AGING_OF_RSEC_SM_LIST
	UPDATE_PRM_SM
	if (PrmM.SA <> LocMasterData.SA)
	UPDATE_SEC_SM
	else
	UPDATE_SEC_SM_FROM_RSEC

4.4.4.4.5 Functions

Table 77 contains the functions used by BMA.

Table 77 – Functions used by BMA

Name	Meaning
BMA_Sync_Ind	The BMA receives a synchronization message from the SRX.
BMA_Announce_Ind	The BMA receives an announce message from the SRX

4.4.4.4.5.1 BMA_Sync_Ind

This service shall be used to receive a synchronization message as shown in Table 78.

Table 78 – BMA_Sync_Ind

Parameter name	Ind
Argument	M
SyncID	M
SM_SA	M
Subdomain	M
Class	M
Stratum	M
Variance	M

Argument
The argument shall convey the specific parameters of the indication.

SyncID
This is the key attribute to identify the instance of the protocol machine.

Attribute Type: Unsigned8

SM_SA
This parameter shall be used for the source MAC address of the synchronization manager. This field shall be coded as data type OctetString[6].

Subdomain
This attribute shall contain the unique identifier of the PTCP subdomain.

Attribute Type: UUID

Class
This attribute shall contain the value for the synchronization class.

Attribute Type: Unsigned8

Stratum
This attribute shall describe the quality of the time issued by a master.

Attribute Type: Unsigned8

Variance
This attribute shall characterize the quality of a clock.

Attribute Type: Integer16

4.4.4.4.5.2 BMA_Announce_Ind

This service shall be used to receive an announce message as shown in Table 79.

Table 79 – BMA_Announce_Ind

Parameter name	Ind
Argument	M
SyncID	M
SM_SA	M
Subdomain	M
Class	M
Stratum	M
Variance	M

Argument
The argument shall convey the specific parameters of the indication.

SyncID
This is the key attribute to identify the instance of the protocol machine.

Attribute Type: Unsigned8

SM_SA
This parameter shall be used for the source MAC address of the synchronization manager. This field shall be coded as data type OctetString[6].

Subdomain
This attribute shall contain the unique identifier of the PTCP subdomain.

Attribute Type: UUID

Class
This attribute shall contain the value of the synchronization class.

Attribute Type: Unsigned8

Stratum
This attribute shall describe the quality of the time issued by a master.

Attribute Type: Unsigned8

Variance
This attribute shall characterize the quality of a clock.

Attribute Type: Integer16

4.4.4.5 Master-Slave-Manager Protocol Machine (MSM)

4.4.4.5.1 Primitive definitions

4.4.4.5.1.1 Primitives exchanged between MSM and ASE

The service primitives including their associated parameters issued by MSM user received by MSM and vice versa are described in the PTCP ASE in the service definition.

4.4.4.5.2 State machine description

The Master-Slave-Manager controls the transition between the roles slave, secondary master and primary master. The transitions are made local by every device for every existing PTCP SyncID. Therefore an instance of the state machine exists for every PTCP SyncID.To achive these transitions the MSM has to inidiate synchronization and announce messages. Announce messages shall be send to advertise master capabilities and grand that only one primary master is active. Synchronization messages shall only send by the primary master.

After start up the data management shall be initialized (POWER-ON) and the synchronization role has to be set (UNKNOWN). The role is set by PTCP services. After this the MSM goes from state UNKNOWN to state IDLE. With primary/secondary master role the MSM will send an announce message bevor it goes to the state IDLE_SEC_M. This behaviour assures that first master capabilities will be announced. A MSM with slave role leaves the state IDLE and goes to the state AS_SLAVE (asynchronous slave), if it receives a synchronization message.

A slave (role == slave) should be easy to implement. Therefore it waits for synchronization messages (AS_SLAVE), synchronize to an active primary master and goes to the state SLAVE if it is synchronous. This implies that only one active primary exists. Otherwise the synchronization will corrupted by the disorientation of the slave.

Multiple active primary masters shall prevented by the state IDLE_SEC_M. With the role primary master, the MSM sends several announce messages at the state IDLE_SEC_M. Therefore the decision to go from state IDLE_SEC_M to state PRM_M (primary manager) will be made with extensive knowledge of concruent masters.

If an active primary master (with fewer capabilities) already exists, a new primary master needs to synchronize to the active master before it can be primary master. For synchronization the MSM goes to state AS_SLAVE, if synchronous to state SEC_M (secondary master) and after several announce messages to state PRM_M.

Like the primary master, a secondary master firstly has to be synchronous. This is meaningful, because an asynchronous secondary manager isn't able to be primary without troubles at slave synchronization. Due to this a synchronous secondary master with bad clock is better than an asynchronous secondary manager with high accurancy clock.

The voting for secondary master starts as soon as a primary master is active. Until then the MSM waits at state IDLE_SEC_M. With a synchronizartion message the state changes from IDLE_SEC_M to AS_SLAVE. At AS_SLAVE several announce messages shall send and synchronity will achieved bevor the MSM goes to state SEC_M.

A burst of announce messages occur before primary master voting and bevor secondary master voting.

Figure 33 shows the MSM state machine:

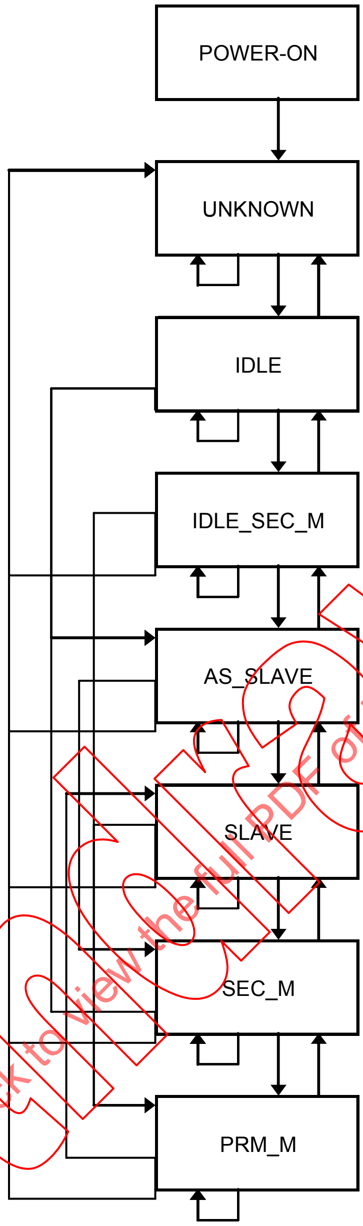

Figure 33 – State transition diagram of MSM

States of the MSM

POWER_ON
Data initialization

UNKNOWN
The synchronization role is unknown. A role shall set via PTCP service.

IDLE
Role == Slave: Wait for synchronization messages.
Role == Primary/Secondary Master: Send announce messages.

IDLE_SEC_M
Role == Primary Master: Send several announce message to be sure that this device should be primary master.

AS_SLAVE
Synchronize to the active master, before a transition to slave, primary master or secondary master will be done.

SLAVE
The MSM has a slave role and achieved synchronity with the primary master.

SEC_M
This is the active secondary master. Therefore it sends announce messages.

PRM_M
This is the active primary master. Therefore it sends synchronization messages.

Local variables of the BMA

DestRole (Unsigned16)
This local variable contains the destination role for synchronization.

4.4.4.5.3 MSM state table

Table 80 contains the state table used by MSM.

Table 80 – MSM state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> DestRole := UNKNOWN	UNKNOWN
2	UNKNOWN	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN =>	UNKNOWN
3	UNKNOWN	**MSM_Role_Ind (SyncID, Role)** /Role <> UNKNOWN => DestRole :=Role IdleT.start (SyncID)	IDLE
4	UNKNOWN	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** => ignore	UNKNOWN
5	UNKNOWN	**MSM_Sync_Cnf (SyncID, Status)** => ignore	UNKNOWN
6	UNKNOWN	**MSM_Announce_Cnf (SyncID, Status)** => ignore	UNKNOWN
7	IDLE	**IdleT.expired (SyncID)** /DestRole == UNKNOWN =>	UNKNOWN
8	IDLE	**IdleT.expired (SyncID)** /DestRole == SLAVE => IdleT.start (SyncID)	IDLE
9	IDLE	**IdleT.expired (SyncID)** /DestRole == SEC_M || DestRole == PRM_M => NReturn := 0 OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	IDLE_SEC_M

#	Current State	Event /Condition =>Action	Next State
10	IDLE	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** /DestRole == SLAVE => IdleT.stop (SyncID) M_SA := SM_SA L_Time := LocalTime M_Time := MasterTime OFFSET_CTRL_START (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	AS_SLAVE
11	IDLE	**MSM_Sync_Cnf (SyncID, Status)**=>ignore	AS_SLAVE
12	IDLE	**MSM_Announce_Cnf (SyncID, Status)** => ignore	AS_SLAVE
13	IDLE	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN => IdleT.stop (SyncID) DestRole :=Role	UNKNOWN
14	IDLE	**MSM_Role_Ind (SyncID, Role)** /Role <> UNKNOWN => DestRole := Role	IDLE
15	IDLE_SEC_M	**OffsetContLoopT.expired (SyncID)** /NReturn < MIN_REPEAT_ANNOUNCE => NReturn++ OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	IDLE_SEC_M
16	IDLE_SEC_M	**OffsetContLoopT.expired (SyncID)** /NReturn >= MIN_REPEAT_ANNOUNCE && DestRole == SEC_M => OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	IDLE_SEC_M
17	IDLE_SEC_M	**OffsetContLoopT.expired (SyncID)** /NReturn >= MIN_REPEAT_ANNOUNCE && DestRole == PRM_M => OffsetContLoopT.start (SyncID) BRC_Role_Ind (SyncID, MASTER) MSM_Sync_Req (SyncID)	PRM_M
18	IDLE_SEC_M	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** => M_SA := SM_SA L_Time := LocalTime M_Time := MasterTime OFFSET_CTRL_START (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	AS_SLAVE
19	IDLE_SEC_M	**MSM_Sync_Cnf (SyncID, Status)** => ignore	IDLE_SEC_M
20	IDLE_SEC_M	**MSM_Announce_Cnf (SyncID, Status)** /Status == OK => ignore	IDLE_SEC_M
21	IDLE_SEC_M	**MSM_Announce_Cnf (SyncID, Status)**/Status <> OK=>MSM_Announce_Req (SyncID)	IDLE_SEC_M

#	Current State	Event /Condition =>Action	Next State
22	IDLE_SEC_M	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN => OffsetContLoopT.stop (SyncID) DestRole := Role	UNKNOWN
23	IDLE_SEC_M	**MSM_Role_Ind (SyncID, Role)** /Role == SLAVE => OffsetContLoopT.stop (SyncID) DestRole := Role IdleT.start (SyncID)	IDLE
24	IDLE_SEC_M	**MSM_Role_Ind (SyncID, Role)** /Role == PRM_M || Role == SEC_M => DestRole := Role	IDLE_SEC_M
25	AS_SLAVE	**OffsetContLoopT.expired (SyncID)** /!OFFSET_CTRL_IS_SYNC && Role == SLAVE => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	AS_SLAVE
26	AS_SLAVE	**OffsetContLoopT.expired (SyncID)** /!OFFSET_CTRL_IS_SYNC && (Role == SEC_M || Role == PRM_M) => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	IDLE_SEC_M
27	AS_SLAVE	**OffsetContLoopT.expired (SyncID)** /OFFSET_CTRL_IS_SYNC && Role == SLAVE => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	SLAVE
28	AS_SLAVE	**OffsetContLoopT.expired (SyncID)** /OFFSET_CTRL_IS_SYNC && (Role == SEC_M || Role == PRM_M) => NReturn := 0 OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	SEC_M
29	AS_SLAVE	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** => M_SA == SM_SA L_Time := LocalTime M_Time := MasterTime	AS_SLAVE
30	AS_SLAVE	**MSM_Sync_Cnf (SyncID, Status)** => ignore	AS_SLAVE
31	AS_SLAVE	**MSM_Announce_Cnf (SyncID, Status)** => ignore	AS_SLAVE
32	AS_SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN => DestRole := Role	UNKNOWN
33	AS_SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == SLAVE => DestRole := Role	AS_SLAVE

#	Current State	Event /Condition =>Action	Next State
34	AS_SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == PRM_M || Role == SEC_M => DestRole := Role	AS_SLAVE
35	SLAVE	**OffsetContLoopT.expired (SyncID)** /!OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	AS_SLAVE
36	SLAVE	**OffsetContLoopT.expired (SyncID)** /OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	SLAVE
37	SLAVE	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** => M_SA := SM_SA L_Time := LocalTime M_Time := MasterTime	SLAVE
38	SLAVE	**MSM_Sync_Cnf (SyncID, Status)** => ignore	SLAVE
39	SLAVE	**MSM_Announce_Cnf (SyncID, Status)** => ignore	SLAVE
40	SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN => DestRole := Role OffsetContLoopT.stop (SyncID)	UNKNOWN
41	SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == SLAVE => DestRole := Role	SLAVE
42	SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == SEC_M => NReturn := 0 DestRole := Role	SEC_M
43	SLAVE	**MSM_Role_Ind (SyncID, Role)** /Role == PRM_M => DestRole := Role BRC_Role_Ind (SyncID, MASTER)	PRM_M
44	SEC_M	**OffsetContLoopT.expired (SyncID)** /!OFFSET_CTRL_IS_SYNC => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID)	AS_SLAVE
45	SEC_M	**OffsetContLoopT.expired (SyncID)** /NReturn < MIN_REPEAT_ANNOUNCE && OFFSET_CTRL_IS_SYNC => NReturn++ OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	SEC_M

#	Current State	Event /Condition =>Action	Next State
46	SEC_M	**OffsetContLoopT.expired (SyncID)** /NReturn > MIN_REPEAT_ANNOUNCE && OFFSET_CTRL_IS_SYNC && Role == SEC_M => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID) MSM_Announce_Req (SyncID)	SEC_M
47	SEC_M	**OffsetContLoopT.expired (SyncID)** /NReturn > MIN_REPEAT_ANNOUNCE && OFFSET_CTRL_IS_SYNC && Role == PRM_M => OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) OffsetContLoopT.start (SyncID) BRC_Role_Ind (SyncID, MASTER) MSM_Sync_Req (SyncID)	PRM_M
48	SEC_M	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** => M_SA := SM_SA L_Time := LocalTime M_Time := MasterTime	SEC_M
49	SEC_M	**MSM_Sync_Cnf (SyncID, Status)** => ignore	SEC_M
50	SEC_M	**MSM_Announce_Cnf (SyncID, Status)** /Status == OK => ignore	SEC_M
51	SEC_M	**MSM_Announce_Cnf (SyncID, Status)** /Status <> OK => MSM_Announce_Req (SyncID)	SEC_M
52	SEC_M	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN => DestRole := Role OFFSET_CTRL_STOP (SyncID) OffsetContLoopT.stop (SyncID)	UNKNOWN
53	SEC_M	**MSM_Role_Ind (SyncID, Role)** /Role == SLAVE => DestRole := Role	SLAVE
54	SEC_M	**MSM_Role_Ind (SyncID, Role)** /Role == SEC_M || Role == PRM_M => DestRole := Role	SEC_M
55	PRM_M	**OffsetContLoopT.expired (SyncID)** => OffsetContLoopT.start (SyncID) MSM_Sync_Req (SyncID)	PRM_M
56	PRM_M	**MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)** => ignore	PRM_M
57	PRM_M	**MSM_Sync_Cnf (SyncID, Status)** /Status == OK => ignore	PRM_M

#	Current State	Event /Condition =>Action	Next State
58	PRM_M	**MSM_Sync_Cnf (SyncID, Status)** /Status <> OK => ignore MSM_Sync_Req (SyncID)	PRM_M
59	PRM_M	**MSM_Announce_Cnf (SyncID, Status)** => ignore	PRM_M
60	PRM_M	**MSM_Role_Ind (SyncID, Role)** /Role == UNKNOWN => OffsetContLoopT.stop (SyncID) DestRole := Role	UNKNOWN
61	PRM_M	**MSM_Role_Ind (SyncID, Role)** /Role == SLAVE => DestRole := Role BRC_Role_Ind (SyncID, SLAVE)	SLAVE
62	PRM_M	**MSM_Role_Ind (SyncID, Role)** /Role == SEC_M => NReturn:= 0 DestRole := Role BRC_Role_Ind (SyncID, SLAVE) MSM_Announce_Req (SyncID, SYNC)	SEC_M
63	PRM_M	**MSM_Role_Ind (SyncID, Role)** /Role == PRM_M => DestRole := Role	PRM_M

4.4.4.5.4 Macros

Table 81 contains the macros used by MSM.

Table 81 – Macros used by MSM

Name	Meaning
OFFSET_CTRL_START (SyncID, M_SA, L_Time, M_Time)	Initialize control loop for offset compenstation
OFFSET_CTRL_CALC (SyncID, M_SA, L_Time, M_Time)	Offset compensaton
OFFSET_CTRL_STOP (SyncID)	Stop offset compensation
OFFSET_CTRL_IS_SYNC	Check PLL Window of control loop for offset compensation

4.4.4.5.5 Functions

Table 82 contains the functions used by MSM.

Table 82 – Functions used by MSM

Name	Meaning
MSM_Role_Ind (SyncID, Role)	The MSM receives his role for synchronization.
MSM_Sync_Req (SyncID)	The MSM issues a synchronization messages. Create PTCP-PDU according PTCP_RTASyncPDU Assignments: D_Port := AUTO DA :=PTCP_MulticastMACadd for PTCP_RTASyncPDU and corresponding PTCP_SyncID SA := local source address A_SDU := LT, FrameID, PTCP_RTASyncPDU LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VALN_Prio, VLANID, A_SDU)
MSM_Sync_Cnf (SyncID, Status)	The MSM gets a confirmation for one send synchronization message. LMPM_A_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status
MSM_Announce_Req (SyncID)	The MSM issues an annonce message. Create PTCP-PDU according PTCP_AnnouncePDU Assignments: D_Port := AUTO DA :=PTCP_MulticastMACadd for PTCP_AnnouncePDU and corresponding PTCP_SyncID SA := local source address A_SDU := LT, FrameID, PTCP_RTASyncPDU LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VALN_Prio, VLANID, A_SDU)
MSM_Announce_Cnf (SyncID, Status)	The MSM gets a confirmation for one send announce message. LMPM_A_Data.conf (CREP, D_Port, TStamp, LMPM_status) Assignments: Status := LMPM_status

4.4.4.5.5.1 MSM_Role_Ind

This service shall be used to indicate a new role to the MSM as shown in Table 83.

Table 83 – MSM_Role_Ind

Parameter name	Ind
Argument	M
SyncID	M
Role	M

Argument
The argument shall convey the specific parameters of the indication.

SyncID
This ist the key attribute to identify the instance of the protocol machine.

Attribute Type: Unsigned8

Role
This attribute specifies the role for synchronization. It may be set to the following values:

- SLAVE: This value activates slave functionality.
- SEC_SM: This value activates secondary master functionality.
- PRM_SM: This value activates primary master functionality.

Attribute Type: Unsigned16

4.4.4.5.5.2 MSM_Sync

This service shall be used to send a synchronization message as shown Table 84.

Table 84 – MSM_Sync

Parameter name	Req	Cnf
Argument	M	
SyncID	M	
Result		M
SyncID		M(=)
Status		M

Argument
The argument shall convey the specific parameters of the indication.

SyncID
This ist the key attribute to identify the instance of the protocol machine.

Attribute Type: Unsigned8

Result
This parameter indicates the success of the service request succeeded.

SyncID
The same attribute as in argument.

Status
This attribute contains the returned status of the LMPM-Function.

Attribute Type: Unsigned32

4.4.4.5.5.3 MSM_Announce

This service shall be used to send an announce message as shown in Table 85.

Table 85 – MSM_Announce

Parameter name	Req	Cnf
Argument	M	
SyncID	M	
Result		M
SyncID		M(=)
Status		M

Argument
The argument shall convey the specific parameters of the indication.

SyncID
This ist the key attribute to identify the instance of the protocol machine.

Attribute Type: Unsigned8

Result
This parameter indicates the success of the service request succeeded.

SyncID
The same attribute as in argument.

Status
This attribute contains the returned status of the LMPM-Function.

4.4.4.6 Bridge-Rate-Control Protocol Machine (BRC)

4.4.4.6.1 Primitive definitions

4.4.4.6.1.1 Primitives exchanged between BRC and ASE

4.4.4.6.1.2 Parameters of BRC primitives

The service primitives including their associated parameters issued by BRC user received by BRC and vice versa are described in the PTCP ASE in the service definition.

4.4.4.6.2 State machine description

The Bridge-Rate-Control state machine shall start the rate control or shall it slowy shutdown for a PTCP-Bridge. A standard bridge will remain at the state UNKNOWN. Rate control needs to be slowly shutdown, if a slave gets the master role. For a slave the rate control is necessary. Therefore it shall be started with the transfer from IDLE to SLAVE state.

Figure 34 shows the BRC state machine:

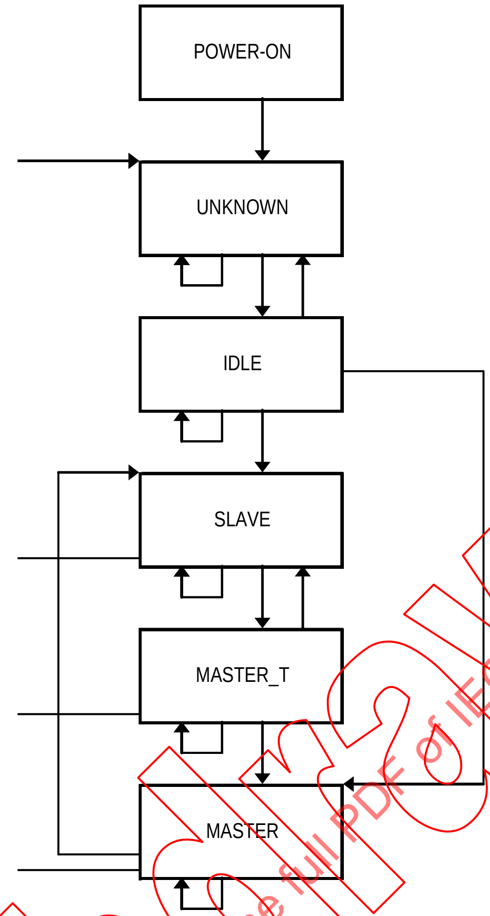

Figure 34 – State transition diagram of BRC

States of the BRC

POWER_ON
Data initialization

UNKNOWN
This state indicates a bridge without PTCP (a bridge without rate control) or a single-port-device. The state machine goes from state UNKNOWN to state IDLE if the PTCP-Bridge is started (PTCP_Start_Bridge.req).

IDLE
At this state the PTCP-Bridge waites for a role indication. If the state machine has the role slave, it will start the rate control with the first received synchronization message. As master the state will change to MASTER. A master needs no rate control.

SLAVE
PTCP-Bridge with active rate-control

MASTER_T
Tis is a transient state from slave to master. If a slave gets the master role, it shall slowly reduce his rate control to eliminate rate corrections.

MASTER
PTCP-Bridge with rate-control-factor == 1. This means, the master-clock needs no correction.

Local variables of the BRC

RateCompensationFactor (Unsigned32)
This local variable contains the rate compensation factor of a slave.

4.4.4.6.3 BRC state table

Table 86 contains the state table used by BRC.

Table 86 – BRC state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> RateCompensationFactor := 1	UNKNOWN
2	UNKNOWN	**RateContLoopT.expired (SyncID)** => ignore	UNKNOWN
3	UNKNOWN	**BRC_Sync_Ind (SyncID, SM_SA, LocalTime, MasterTime)** => ignore	UNKNOWN
4	UNKNOWN	**BRC_Role_Ind (SyncID, Role)** => ignore	UNKNOWN
5	UNKNOWN	**PTCP_Start_Bridge.req (SyncID, PortParameter)** => RateCompensationFactor := 1 Receipt := 0 ErrCode := OK PTCP_Start_Bridge.cnf (SyncID, ErrCode)	IDLE
6	UNKNOWN	**PTCP_Stop_Bridge.req (SyncID)** => ErrCode := NOT_STARTED PTCP_Stop_Bridge.cnf (SyncID, ErrCode)	UNKNOWN
7	IDLE	**RateContLoopT.expired (SyncID)** => ignore	IDLE
8	IDLE	**BRC_Sync_Ind (SyncID, SM_SA, LocalTime, MasterTime)** => M_SA := SM_SA L_Time := LocalTime M_Time := MasterTime RATE_CTRL_START (SyncID, M_SA, L_Time, M_Time) RateContLoopT.start (SyncID)	SLAVE
9	IDLE	**BRC_Role_Ind (SyncID, Role)** /Role == MASTER =>	MASTER
10	IDLE	**BRC_Role_Ind (SyncID, Role)** /Role == SLAVE => ignore	IDLE
11	IDLE	**PTCP_Start_Bridge.req (SyncID, PortParameter)** => ErrCode := STARTED PTCP_Start_Bridge.cnf (SyncID, ErrCode)	IDLE
12	IDLE	**PTCP_Stop_Bridge.req (SyncID)** => ErrCode := OK PTCP_Stop_Bridge.cnf (SyncID, ErrCode)	UNKNOWN
13	SLAVE	**RateContLoopT.expired (SyncID)**=>RATE_CTRL_CALC (SyncID, M_SA, L_Time, M_Time) RateContLoopT.start (SyncID)	SLAVE
14	SLAVE	**BRC_Sync_Ind (SyncID, SM_SA, LocalTime, MasterTime)** => M_SA := SM_SA L_Time := LocalTime M_Time := MasterTime	SLAVE

#	Current State	Event /Condition =>Action	Next State
15	SLAVE	**BRC_Role_Ind (SyncID, Role)** /Role == SLAVE => ignore	SLAVE
16	SLAVE	**BRC_Role_Ind (SyncID, Role)** /Role == MASTER =>	MASTER_T
17	SLAVE	**PTCP_Start_Bridge.req (SyncID, PortParameter)** => ErrCode := STARTED PTCP_Start_Bridge.cnf (SyncID, ErrCode)	SLAVE
18	SLAVE	**PTCP_Stop_Bridge.req (SyncID)** => ErrCode := OK PTCP_Stop_Bridge.cnf (SyncID, ErrCode)	UNKNOWN
19	MASTER_T	**RateContLoopT.expired (SyncID)** /!CHECK_RATE_DECAYED => RATE_CTRL_DECAY_CALC (SyncID) RateContLoopT.start (SyncID)	MASTER_T
20	MASTER_T	**RateContLoopT.expired (SyncID)** /CHECK_RATE_DECAYED =>	MASTER
21	MASTER_T	**BRC_Sync_Ind (SyncID, SM_SA, LocalTime, MasterTime)** => ignore	MASTER_T
22	MASTER_T	**BRC_Role_Ind (SyncID, Role)** /Role == SLAVE =>	SLAVE
23	MASTER_T	**BRC_Role_Ind (SyncID, Role)** /Role == MASTER => ignore	MASTER_T
24	MASTER_T	**PTCP_Start_Bridge.req (SyncID, PortParameter)** => ErrCode := STARTED PTCP_Start_Bridge.cnf (SyncID, ErrCode)	MASTER_T
25	MASTER_T	**PTCP_Stop_Bridge.req (SyncID)**=>ErrCode := OKPTCP_Stop_Bridge.cnf (SyncID, ErrCode)	UNKNOWN
26	MASTER	**RateContLoopT.expired (SyncID)** => ignore	MASTER
27	MASTER	**BRC_Sync_Ind (SyncID, SM_SA, LocalTime, MasterTime)** => ignore	MASTER
28	MASTER	**BRC_Role_Ind (SyncID, Role)** /Role == SLAVE =>	SLAVE
29	MASTER	**BRC_Role_Ind (SyncID, Role)** /Role == MASTER => ignore	MASTER
30	MASTER	**PTCP_Start_Bridge.req (SyncID, PortParameter)** => ErrCode := STARTED PTCP_Start_Bridge.cnf (SyncID, ErrCode)	MASTER
31	MASTER	**PTCP_Stop_Bridge.req (SyncID)** => ErrCode := OK PTCP_Stop_Bridge.cnf (SyncID, ErrCode)	UNKNOWN

4.4.4.6.4 Macros

Table 87 contains the macros used by BRC.

Table 87 – Macros used by BRC

Name	Function
RATE_CTRL_START (SA, L_Time, M_Time)	Start calculation of RateCompensationFactor
RATE_CTRL_CALC (SA, L_Time, M_Time)	Calculation of RateCompensationFactor
RATE_CTRL_DECAY_CALC	Decay RateCompensationFactor to 1
CHECK_RATE_DECAYED	Check if RateCompensationFactor has the value 1

4.4.4.6.5 Functions

Table 88 contains the functions used by the BRC.

Table 88 – Macros used by the BRC

Name	Function
BRC_Sync_Ind (SyncID, SM_SA, LocalTime, MasterTime)	The BMA receives a synchronization message from the SRX.
BRC_Role_Ind (SyncID, Role)	The BMA receives his role from the MSM.

4.4.4.6.5.1 BRC_Sync_Ind

This service shall be used to receive a synchronization message as shown in Table 89.

Table 89 – BRC_Sync_Ind

Parameter name	Ind
Argument	M
SyncID	M
SM_SA	M
LocalTime	M
MasterTime	M

Argument
The argument shall convey the specific parameters of the indication.

SyncID
This ist the key attribute to identify the instance of the protocol machine.

Attribute Type: Unsigned8

SM_SA
This parameter shall be used for the source MAC address of the synchronization manager. This field shall be coded as data type OctetString[6].

LocalTime
This attribute shall contain the time-stamp at arrival of the synchronization message.

Attribute Type: Unsigned64

MasterTime
This attribute shall contain the time from the active primary master.

Attribute Type: Unsigned64

4.4.4.7 Synchronization-Receive Protocol Machine (SRX)

4.4.4.7.1 State machine description

The SRX state machine hides the follow-up mechanism from upper objects. BRC, MSM and BMA receive only synchronization messages, even if follow-up messages are necessary. In this case the state machine waits for the follow-up and uses this delay values for sync-indication (BRC_Sync_Ind, MSM_Sync_Ind, BMA_Sync_Ind).

Figure 35 shows the SRX state machine:

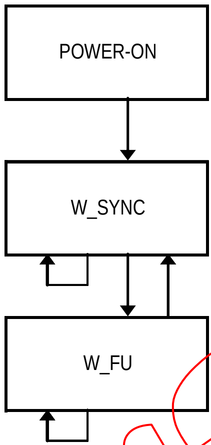

Figure 35 – State transition diagram of SRX

States of the SRX

POWER_ON
Data initialization

W_SYNC
The SRX state machine waits for synchronization messages.

W_FU
The SRX state machine waits for follow-up messages.

4.4.4.7.2 SRX state table

Table 90 contains the state table used by SRX.

Table 90 – SRX state table

#	Current State	Event /Condition =>Action	Next State
1	POWER-ON	=> inialize SRX object	W_SYNC
2	W_SYNC	**SRX_Sync_Ind (SyncID, Fu, S_Port, TStamp, PTCP_PDU)** /!Fu => SM_SA := PTCP_MasterSourceAddress Subdomain := PTCP_SubdomainUUID Class := PTCP_ClockRole Stratum := PTCP_ClockStratum Variance := PTCP_ClockVariance LocalTime := TStamp MasterTime := PTCP_Time + PTCP_Delay BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance) MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime) BRC_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)	W_SYNC
3	W_SYNC	**SRX_Sync_Ind (SyncID, Fu, S_Port, TStamp, PTCP_PDU)** /Fu => SequneceID := PTCP_SequenceID SM_SA := SM_SA := PTCP_MasterSourceAddress Subdomain := PTCP_SubdomainUUID Class := PTCP_ClockRole Stratum := PTCP_ClockStratum Variance := PTCP_ClockVariance Delay := PTCP_Delay LocalTime := Tstamp	W_FU
4	W_SYNC	**SRX_FollowUp_Ind (SyncID, Fu, S_Port, PTCP_PDU)** => ignore	W_SYNC
5	W_SYNC	**SRX_Announce_Ind (SyncID, PTCP_PDU)** / => A_SM_SA := SM_SA := PTCP_MasterSourceAddress A_Subdomain := PTCP_SubdomainUUID A_Class := PTCP_ClockRole A_Stratum := PTCP_ClockStratum A_Variance := PTCP_ClockVariance BMA_Announce_Ind (SyncID, A_SM_SA, A_Subdomain, A_Class, A_Stratum, A_Variance)	W_SYNC
6	W_FU	**SRX_FollowUp_Ind (SyncID, Fu, S_Port, PTCP_PDU)** /SM_SA <> PTCP_MasterSourceAddress => ignore	W_FU
7	W_FU	**SRX_FollowUp_Ind (SyncID, Fu, S_Port, PTCP_PDU)**/SM_SA == PTCP_MasterSourceAddress&& PTCP_SequenceID <> SequneceID=>ignore	W_FU
8	W_FU	**SRX_FollowUp_Ind (SyncID, Fu, S_Port, PTCP_PDU)** /SM_SA == PTCP_MasterSourceAddress && PTCP_SequenceID == SequneceID => Delay := Delay + PTCP_Delay MasterTime := PTCP_Time + Delay BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance) MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime) BRC_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)	W_SYNC

#	Current State	Event /Condition =>Action	Next State
9	W_FU	**SRX_Sync_Ind (SyncID, Fu, S_Port, TStamp, PTCP_PDU)** /Fu => SequneceID := PTCP_SequenceID SM_SA := SM_SA := PTCP_MasterSourceAddress Subdomain := PTCP_SubdomainUUID Class := PTCP_ClockRole Stratum := PTCP_ClockStratum Variance := PTCP_ClockVariance Delay := PTCP_Delay LocalTime := Tstamp	W_FU
10	W_FU	**SRX_Sync_Ind (SyncID, Fu, S_Port, TStamp, PTCP_PDU)** /!Fu => SM_SA := PTCP_MasterSourceAddress Subdomain := PTCP_SubdomainUUID Class := PTCP_ClockRole Stratum := PTCP_ClockStratum Variance := PTCP_ClockVariance LocalTime := TStamp MasterTime := PTCP_Time + PTCP_Delay BMA_Sync_Ind (SyncID, SM_SA, Subdomain, Class, Stratum, Variance) MSM_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime) BRC_Sync_Ind (SyncID, SM_SA, Subdomain, LocalTime, MasterTime)	W_SYNC
11	W_FU	**SRX_Announce_Ind (SyncID, PTCP_PDU)** / => A_SM_SA := SM_SA := PTCP_MasterSourceAddress A_Subdomain := PTCP_SubdomainUUID A_Class := PTCP_ClockRole A_Stratum := PTCP_ClockStratum A_Variance := PTCP_ClockVariance BMA_Announce_Ind (SyncID, A_SM_SA, A_Subdomain, A_Class, A_Stratum, A_Variance)	W_FU

4.4.4.7.3 Functions

Table 91 contains the functions used by SRX.

Table 91 – Functions used by SRX

Name	Function
SRX_Sync_Ind (SyncID, Fu, S_Port, TStamp, PTCP_PDU)	Receive PTCP-PDU according PTCP-RTASyncPDU LMPM_A_Data.ind (S_Port, TSamp, DA, SA, VLANPrio, VLANID, A_SDU) Assignments: SyncID := A_SDU.FrameID & SYNC_ID_MASK Fu := A_SDU.FrameID & FOLLOW_UP_MASK PTCP_PDU := A_SDU whitout LT and FrameID
SRX_FollowUp_Ind (SyncID, Fu, S_Port, PTCP_PDU)	Receive PTCP-PDU according PTCP-FollowUpPDU LMPM_A_Data.ind (S_Port, TSamp, DA, SA, VLANPrio, VLANID, A_SDU) Assignments: SyncID := A_SDU.FrameID & SYNC_ID_MASK PTCP_PDU := A_SDU
SRX_Announce_Ind (SyncID, PTCP_PDU)	Receive PTCP-PDU according PTCP-AnnouncePDU LMPM_A_Data.ind (S_Port, TSamp, DA, SA, VLANPrio, VLANID, A_SDU) Assignments: SyncID := A_SDU.FrameID & SYNC_ID_MASK PTCP_PDU := A_SDU

4.5 Media redundancy

4.5.1 FAL syntax description for MRP

4.5.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.5.1.2 APDU abstract syntax

Table 92 defines the abstract syntax of the MRP-PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 92 – MRP APDU syntax

APDU name	APDU structure
MRP-PDU	MRP_Version (=0x0001), MRP_Type, MRP_Common, [MRP_Option], MRP_End, [Padding*] [a]

[a] If the frame is shorter than 64 octets it will be extended with padding to 64 octets.

Table 93 defines structures for substitutions of elements of the APDU structures shown in Table 92.

Table 93 – MRP substitutions

Substitution name	Structure
MRP_Type	MRP_Test ^ MRP_LinkChange ^ MRP_TopologyChange ^ MRP_Option
MRP_Common	MRP_TLVHeader, MRP_SequenceID, MRP_DomainUUID
MRP_Option	MRP_TLVHeader, MRP_ManufacturerOUI, Data*, [Padding*] [a]
MRP_End	MRP_TLVHeader (=0x0000)
MRP_Test	MRP_TLVHeader, MRP_Prio, MRP_SA [b], MRP_PortRole, MRP_RingState, MRP_Transition, MRP_TimeStamp, [Padding*] [a]
MRP_TopologyChange	MRP_TLVHeader, MRP_Prio, MRP_SA [b], MRP_Interval, [Padding*] [a]
MRP_LinkChange	MRP_LinkUp ^ MRP_LinkDown
MRP_LinkDown	MRP_TLVHeader, MRP_SA [b], MRP_PortRole, MRP_Interval, MRP_Blocked, [Padding*] [a]
MRP_LinkUp	MRP_TLVHeader, MRP_SA [b], MRP_PortRole, MRP_Interval, MRP_Blocked, [Padding*] [a]

[a] 32bit alignment shall be ensured.

[b] Shall be the interface MAC address.

4.5.1.3 Coding of the field MRP_TLVHeader

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: MRP_TLVHeader.Length
The value contains the sum of subsequent octets of the according block.

Bit 8 – 15: MRP_TLVHeader.Type
This field shall be coded with the values according to Table 94.

Table 94 – MRP_TLVHeader.Type

Value (hexadecimal)	Meaning	Usage
0x00	MRP_End (MRP_TLVHeader.Length shall be set to zero)	Mandatory
0x01	MRP_Common	Mandatory
0x02	MRP_Test	Mandatory
0x03	MRP_TopologyChange	Mandatory
0x04	MRP_LinkDown	Mandatory
0x05	MRP_LinkUp	Mandatory
0x06 – 0x7E	Reserved	
0x7F	MRP_Option (Organizationally specific)	Optional

4.5.1.4 Coding of the field MRP_Version

This field shall be coded as data type Unsigned16. This field shall be set to one.

4.5.1.5 Coding of the field MRP_SequenceID

This field shall be coded as Unsigned16. It is used to identify the duplication of MRP frames in the ring. The range is from 0 to 65 535. The requesting application process shall provide an unique sequence number to each outstanding service request.

4.5.1.6 Coding of the field MRP_SA

This field shall be coded as data type OctetString[6]. The value of the field MRP_SA shall be a MAC address according to IEEE 802.

4.5.1.7 Coding of the field MRP_Prio

This field shall be coded as Unsigned16 and set according to Table 95.

Table 95 – MRP_Prio

Value (hexadecimal)	Meaning
0x0000	Highest priority redundancy manager
0x1000 – 0x7000	High priorities
0x8000	Default priority for redundancy manager
0x9000 – 0xE000	Low priorities
0xF000	Lowest priority redundancy manager
other	Reserved

4.5.1.8 Coding of the field MRP_PortRole

This field shall be coded as data type Unsigned16. The coding shall be according to Table 96.

Table 96 – MRP_PortRole

Value (hexadecimal)	Meaning	Usage
0x0000	Primary ring port	Frame is sent on primary ring port
0x0001	Secondary ring port	Frame is sent on backup ring port
0x0002 – 0xFFFF	Reserved	

4.5.1.9 Coding of the field MRP_RingState

This field shall be coded as Unsigned16 with the values according to Table 97.

Table 97 – MRP_RingState

Value (hexadecimal)	Meaning	Usage
0x0000	Ring open	Redundancy manager in Ring open state
0x0001	Ring closed	Redundancy manager in Ring closed state
0x0002 – 0xFFFF	Reserved	

4.5.1.10 Coding of the field MRP_Interval

This field shall be coded as Unsigned16 with the values according to Table 98.

Table 98 – MRP_Interval

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x07D0	Interval for next topologie change event (in ms)	Mandatory
0x07D1 – 0xFFFF	Interval for next topologie change event (in ms)	Optional

4.5.1.11 Coding of the field MRP_Transition

This field shall be coded as Unsigned16 with the values according to Table 99.

Table 99 – MRP_Transition

Value (hexadecimal)	Meaning	Usage
0x0000 – 0xFFFF	Number of transitions from media redundancy ok to media redundancy lost	Used for monitoring this value via a packet sniffer station

4.5.1.12 Coding of the field MRP_TimeStamp

This field shall be coded as Unsigned32 with the values according to Table 100.

Table 100 – MRP_TimeStamp

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0xFFFFFFFF	Actual counter value of 1 ms counter	The value is used by the MRM to determine the maximum travel time of the test frames in a ring

4.5.1.13 Coding of the field MRP_Blocked

This field shall be coded as data type Unsigned16. This field shall be set to one.

4.5.1.14 Coding of the field MRP_ManufacturerOUI

This field shall be coded as OctetString[3] with the Organizationally unique identifier (OUI) as defined by IEEE 802.

Note The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <http://standard.ieee.org/regauth>

4.5.1.15 Coding of the field MRP_DomainUUID

This field shall be coded as UUID with the values according to Table 101.

Table 101 – MRP_DomainUUID

Value (UUID)	Meaning	Usage
00000000-0000-0000-0000-000000000000		Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFE	Unique UUID for MRP redundancy domain	Optional
FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	Default UUID for MRP redundancy domain	Mandatory

4.5.1.16 Coding of the field MRP_LengthDomainName

This field shall be coded as data type Unsigned8.

4.5.1.17 Coding of the field MRP_DomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.32.1.

NOTE The field MRP_DomainName is not terminated by zero.

4.5.2 FAL syntax description for MRRT

4.5.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.5.2.2 APDU abstract syntax

Table 92 defines the abstract syntax of the MRRT-PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 102 – MRRT APDU syntax

APDU name	APDU structure
MRRT-PDU	MRRT_Version(1), MRRT_Type, MRRT_Common, [MRRT_Option], MRRT_End, [Padding*] [a]

[a] If the frame is shorter than 64 octets it will be extended with padding to 64 octets.

Table 93 defines structures for substitutions of elements of the APDU structures shown in Table 92.

Table 103 – MRRT substitutions

Substitution name	Structure
MRRT_Type	MRRT_Test ^ MRRT_Option
MRRT_Common	MRRT_TLVHeader, MRRT_SequenceID, MRRT_DomainUUID
MRRT_Option	MRRT_TLVHeader, MRRT_ManufacturerOUI, Data*, [Padding*] [a]
MRRT_End	MRRT_TLVHeader (=0x0000)
MRRT_Test	MRRT_TLVHeader, MRRT_SA [b], [Padding*] [a]

[a] 32bit alignment shall be ensured.

[b] Shall be the interface MAC address.

4.5.2.3 Coding of the field MRRT_TLVHeader

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: MRRT_TLVHeader.Length
The value contains the sum of subsequent octets of the according block.

Bit 8 – 15: MRRT_TLVHeader.Type
This field shall be coded with the values according to Table 94.

Table 104 – MRRT_TLVHeader.Type

Value (hexadecimal)	Meaning	Usage
0x00	MRRT_End (MRRT_TLVHeader.Length shall be set to zero)	Mandatory
0x01	MRRT_Common	Mandatory
0x02	MRRT_Test	Mandatory
0x03 – 0x7E	Reserved	
0x7F	MRRT_Option (Organizationally Specific)	Optional

4.5.2.4 Coding of the field MRRT_Version

This field shall be coded as data type Unsigned16. This field shall be set to one.

4.5.2.5 Coding of the field MRRT_SequenceID

This field shall be coded as Unsigned16. It is used to identify the duplication of MRRT frames in the ring. The range is from 0 to 65 535. The requesting application process shall provide an unique sequence number to each outstanding service request.

4.5.2.6 Coding of the field MRRT_SA

This field shall be coded as data type OctetString[6]. The value of the field MRRT_SA shall be according to IEEE 802 MAC address.

4.5.2.7 Coding of the field MRRT_ManufacturerOUI

This field shall be coded as OctetString[3] with the Organizationally unique identifier (OUI) as defined by IEEE 802.

Note The value of this central administrative number is given by the IEEE Registration Authority Committee. Available at <http://standard.ieee.org/regauth>.

4.5.2.8 Coding of the field MRRT_DomainUUID

This field shall be coded as UUID according to Table 105.

Table 105 – MRRT_DomainUUID

Value (hexadecimal)	Meaning	Usage
0x00000000-0000-0000-0000-000000000000	Reserved	Reserved
0x00000000-0000-0000-0000-000000000001 – 0xFFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFE	Unique UUID for MRRT redundancy domain	Optional
0xFFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	Default UUID for MRRT redundancy domain	Mandatory

4.5.3 AP-Context state machine

There is no AP-Context State Machine defined for this protocol.

4.5.4 FAL Service Protocol Machines (FSPMs)

4.5.4.1 MRM Protocol Machine for MRP

Figure 36 shows the principal behavior of the protocol machine for a media redundancy manager MRM.

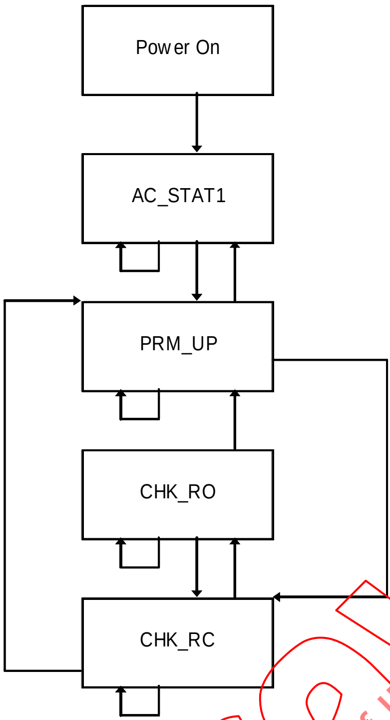

Figure 36 – MRM protocol machine for MRP

The protocol machine states perform in general the following tasks:

- **Power On**
  Initialization, the MRM shall start with both ring ports RPort_1 and RPort_2 in the port state BLOCKED. Static FDB entries for MRP multicast addresses MC_TEST and MC_CONTROL to host are generated. All MRP-PDU have highest priority (ORG).
- **AC_STAT1**
  Startup, waiting for the first Link Up at one of its ring ports (primary ring port), starting test monitoring of the ring and transition to PRM_UP.
- **PRM_UP (Primary Ring Port with Link Up)**
  This state shall be reached if only the primary ring port has a link (secondary ring port with no link). The MRM shall send test frames periodically thru both ring ports.
- **CHK_RO (Check Ring, Ring Open State)**
  The MRM didn't receive its test frames for a determined time, MRP_RingState in ring open state.
- **CHK_RC (Check Ring, Ring Closed State)**
  The MRM shall send its test frames and shall check the link of its ring ports, MRP_RingState in ring closed state..

Local variables of the MRM protocol machine are listed in Table 106.

Table 106 – Local variables of MRM protocol machine

Name	Type	Meaning
SA_Port1	OctetString[6]	Ring port RPort_1 MAC source address of host
SA_Port2	OctetString[6]	Ring port RPort_2 MAC source address of host
SA_RPort	OctetString[6]	Ring port 1 or ring port 2 MAC source address
PRIORITY	Unsigned8	Priority according to IEEE 802.1Q for MRP-PDU. Shall be set to ORG.
MRP_TS_Prio	Unsigned16	MRP_Prio of host
MRP_TS_SA	OctetString[6]	MAC source address of host
RPort_1	Unsigned16	Port identification of ring port 1
RPort_2	Unsigned16	Port identification of ring port 2
PRM_RPort	Unsigned16	Port identification of primary ring port
SEC_RPort	Unsigned16	Port identification of secondary ring port
MRP_MRM_NRmax	Unsigned16	Maximum retransmission count of MRP_TEST-PDU
MRP_MRM_NReturn	Unsigned16	Counter, Range MRP_MRM_NRmax … 0
TC_NReturn	Unsigned16	Counter, Range MRP_TOPNRmax … 0
AddTest	Boolean	Send additional MRP-PDU of type MRP_Test after MRP_TSTshortT intervall if TRUE
MRP_LNK_UP	Unsigned16	Constant value to indicate Link Up
MRP_LNK_DOWN	Unsigned16	Constant value to indicate Link Down

The MRM state machine shall be according to Table 107.

Table 107 – MRM state machine

#	Current State	Event /Condition =>Action	Next State
1	Power On	 => INIT_FDB ADD_MAC_FDB({local},{MC_TEST, MC_CONTROL},ORG) PRM_RPort := RPort_1 SEC_RPort := RPort_2 MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 AddTest := FALSE Set_Port_State.req (PRM_RPort, BLOCKED) Set_Port_State.req (SEC_RPort, BLOCKED)	AC_STAT1
2	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => Set_Port_State.req (PRM_RPort, FORWARDING) TestRingReq(MRP_TSTdefaultT)	PRM_UP
3	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	AC_STAT1

#	Current State	Event /Condition =>Action	Next State
4	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => SEC_RPort := PRM_RPort PRM_RPort := RPort Set_Port_State.req (PRM_RPort, FORWARDING) TestRingReq(MRP_TSTdefaultT)	PRM_UP
5	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	AC_STAT1
6	AC_STAT1	**TestTimer expired** => ignore	AC_STAT1
7	PRM_UP	**TestTimer expired** => AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	PRM_UP
8	PRM_UP	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PRM_UP
9	PRM_UP	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => TestTimer.stop SetPortState.req(PRM_RPort, BLOCKED)	AC_STAT1
10	PRM_UP	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	PRM_UP
11	PRM_UP	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 TestRingReq(MRP_TSTdefaultT)	CHK_RC

#	Current State	Event /Condition =>Action	Next State
12	PRM_UP	**TestRingInd(MRP_SA, MRP_Prio)** /MRP_SA == MRP_TS_SA => MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 TestRingReq(MRP_TSTdefaultT)	CHK_RC
13	PRM_UP	**TestRingInd(MRP_SA, MRP_Prio)** /MRP_SA != MRP_TS_SA => ignore	PRM_UP
14	PRM_UP	**LinkChangeInd(PortMode, Link_status)** /!AddTest => AddTest := TRUE TestRingReq(MRP_TSTshortT)	PRM_UP
15	PRM_UP	**LinkChangeInd(PortMode, Link_status)** /AddTest => ignore	PRM_UP
16	PRM_UP	**TopologyChangeInd(MRP_SA, t)** => ignore	PRM_UP
17	CHK_RO	**TestTimer expired** => AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	CHK_RO
18	CHK_RO	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RO
19	CHK_RO	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => PRM_RPort := SEC_RPort SEC_RPort := RPort TestRingReq(MRP_TSTdefaultT) TopologyChangeReq SetPortState.req(PRM_RPort, BLOCKED)	PRM_UP
20	CHK_RO	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RO

#	Current State	Event /Condition =>Action	Next State
21	CHK_RO	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => SetPortState.req(PRM_RPort, BLOCKED)	PRM_UP
22	CHK_RO	**TestRingInd(MRP_SA, MRP_Prio)** /MRP_SA == MRP_TS_SA => Set_Port_State.req (SEC_RPort, BLOCKED) MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 TestRingReq(MRP_TSTdefaultT) TopologyChangeReq(MRP_TOPchgT)	CHK_RC
23	CHK_RO	**TestRingInd(MRP_SA, MRP_Prio)** /MRP_SA != MRP_TS_SA => ignore	CHK_RO
24	CHK_RO	**LinkChangeInd(PortMode, Link_status)** /!AddTest => AddTest := TRUE TestRingReq(MRP_TSTshortT)	CHK_RO
25	CHK_RO	**LinkChangeInd(PortMode, Link_status)** /AddTest => ignore	CHK_RO
26	CHK_RO	**TopologyChangeInd(MRP_SA, t)** => ignore	CHK_RO
27	CHK_RC	**TestTimer expired** /MRP_MRM_NReturn >= MRP_MRM_NRmax => Set_Port_State.req (SEC_RPort, FORWARDING) MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0 AddTest := FALSE TopologyChangeReq TestRingReq(MRP_TSTdefaultT)	CHK_RO
28	CHK_RC	**TestTimer expired** /MRP_MRM_NReturn < MRP_MRM_NRmax => MRP_MRM_NReturn := MRP_MRM_NReturn + 1 AddTest := FALSE TestRingReq(MRP_TSTdefaultT)	CHK_RC
29	CHK_RC	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RC

#	Current State	Event /Condition =>Action	Next State
30	CHK_RC	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (SEC_RPort, BLOCKED) Set_Port_State.req (PRM_RPort, FORWARD) TestRingReq(MRP_TSTdefaultT) TopologyChangeReq(MRP_TOPchgT)	PRM_UP
31	CHK_RC	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	CHK_RC
32	CHK_RC	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	PRM_UP
33	CHK_RC	**TestRingInd(MRP_SA, MRP_Prio)** /MRP_SA == MRP_TS_SA => MRP_MRM_NRmax := MRP_TSTNRmax – 1 MRP_MRM_NReturn := 0	CHK_RC
34	CHK_RC	**TestRingInd(MRP_SA, MRP_Prio)** /MRP_SA != MRP_TS_SA => ignore	CHK_RC
35	CHK_RC	**LinkChangeInd(PortMode, Link_status)** /AddTest => ignore	CHK_RC
36	CHK_RC	**LinkChangeInd(PortMode, Link_status)** /!AddTest => AddTest := TRUE TestRingReq(MRP_TSTshortT)	CHK_RC
37	CHK_RC	**TopologyChangeInd(MRP_SA, t)** => ignore	CHK_RC
38	AC_STAT1	**LinkChangeInd(PortMode, Link_status)** => ignore	AC_STAT1

4.5.4.2 MRC Protocol Machine for MRP

Figure 37 shows the principal behavior of the protocol machine for a media redundancy client MRC.

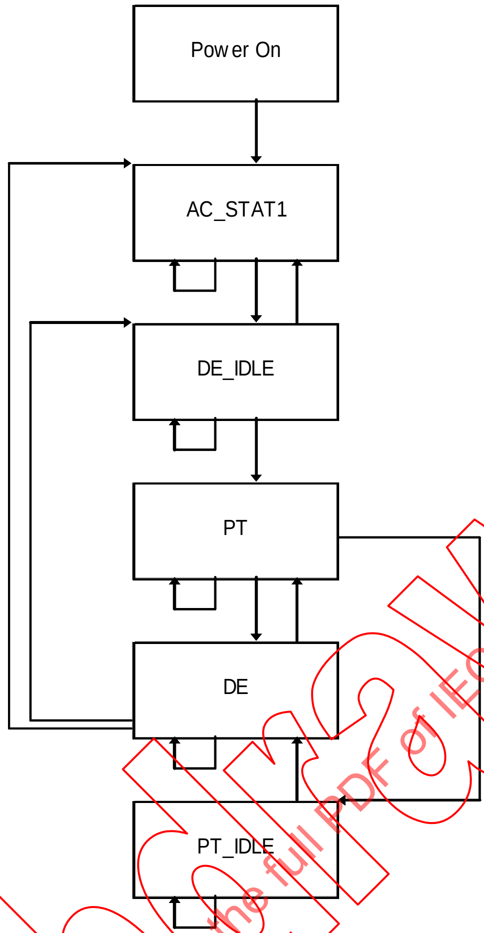

Figure 37 – MRC protocol machine

The protocol machine states perform in general the following tasks:

- **Power On**
  Initialization, the MRC shall start with both ring ports RPort_1 and RPort_2 in the port state BLOCKED. Static FDB entries for MRP multicast addresses MC_TEST and MC_CONTROL are generated: Forward MRP frames to MC_TEST and MC_CONTROL between ring ports and frames to MC_CONTROL also to host. All MRP-PDU have highest priority (ORG).
- **AC_STAT1**
  Startup, wait for Link Up on one of the ring ports.
- **DE_IDLE (Data Exchange idle state)**
  This state shall be reached if only one ring port (primary) has a link and is set to FORWARDING.
- **PT (Pass Through)**
  Temporary state while signalling link changes.
- **DE (Data Exchange)**
  Temporary state while signalling link changes.
- **PT_IDLE (Pass Through idle state)**
  This state shall be reached if both ring ports have a link and are set to FORWARDING.

Local variables of the MRC protocol machine are listed in Table 108.

Table 108 – Local variables of MRC protocol machine

Name	Type	Meaning
SA_RPort	OctetString[6]	Ring port 1 or ring port 2 MAC source address
PRIORITY	Unsigned8	Priority according to IEEE 802.1Q for MRP-PDU. Shall be set to ORG.
RPort_1	Unsigned16	Port identification of ring port 1
RPort_2	Unsigned16	Port identification of ring port 2
PRM_RPort	Unsigned16	Port identification of primary ring port
SEC_RPort	Unsigned16	Port identification of secondary ring port
MRP_LNKNReturn	Unsigned16	Counter, Range MRP_LNKNRmax ... 0
MRP_LNK_UP	Unsigned16	Constant value to indicate Link Up
MRP_LNK_DOWN	Unsigned16	Constant value to indicate Link Down

The MRC state machine shall be according to Table 109.

Table 109 – MRC state machine

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> INIT_FDB ADD_MAC_FDB({RPort_1,RPort_2}, {MC_TEST, MC_CONTROL},ORG) ADD_MAC_FDB({local},{MC_CONTROL},ORG) PRM_RPort := RPort_1 SEC_RPort := RPort_2 Set_Port_State.req (PRM_RPort, BLOCKED) Set_Port_State.req (SEC_RPort, BLOCKED) UpTimer.ini DownTimer.ini	AC_STAT1
2	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => Set_Port_State.req (PRM_RPort, FORWARDING)	DE_IDLE
3	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** / Link_status ==MRP_LNK_DOWN => ignore	AC_STAT1
4	AC_STAT1	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => SEC_RPort := PRM_RPort PRM_RPort := RPort Set_Port_State.req (PRM_RPort, FORWARDING)	DE_IDLE
5	AC_STAT1	**TopologyChangeInd (MRP_SA, t)** => ignore	AC_STAT1

#	Current State	Event /Condition =>Action	Next State
6	DE_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.start(MRP_LNKupT) LinkChangeReq(PRM_RPort, MRP_LNK_UP, MRP_LNKNReturn * MRP_LNKupT)	PT
7	DE_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	DE_IDLE
8	DE_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => Set_Port_State.req (PRM_RPort, BLOCKED)	AC_STAT1
9	DE_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	DE_IDLE
10	DE_IDLE	**TopologyChangeInd(MRP_SA, t)** => CLEAR_FDB(t)	DE_IDLE
11	PT	**UpTimer expired** /MRP_LNKNReturn == 0 => MRP_LNKNReturn := MRP_LNKNRmax Set_Port_State.req (SEC_RPort, FORWARDING)	PT_IDLE
12	PT	**UpTimer expired** /MRP_LNKNReturn > 0 => MRP_LNKNReturn := MRP_LNKNReturn -1 UpTimer.start(MRP_LNKupT) LinkChangeReq(PRM_RPort, MRP_LNK_UP, MRP_LNKNReturn * MRP_LNKupT)	PT
13	PT	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT

#	Current State	Event /Condition =>Action	Next State
14	PT	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.stop Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort , MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE
15	PT	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.stop PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort , MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE
16	PT	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT
17	PT	**TopologyChangeInd(MRP_SA, t)** => MRP_LNKNReturn := MRP_LNKNRmax UpTimer.stop Set_Port_State.req (SEC_RPort, FORWARDING) CLEAR_FDB(t)	PT_IDLE
18	DE	**DownTimer expired** /MRP_LNKNReturn == 0 => MRP_LNKNReturn := MRP_LNKNRmax	DE_IDLE
19	DE	**DownTimer expired** /MRP_LNKNReturn > 0 => MRP_LNKNReturn := MRP_LNKNReturn – 1 DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort, MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE

#	Current State	Event /Condition =>Action	Next State
20	DE	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => MRP_LNKNReturn := MRP_LNKNRmax DownTimer.stop UpTimer.start(MRP_LNKupT) LinkChangeReq(PRM_RPort , MRP_LNK_UP, MRP_LNKNReturn * MRP_LNKupT)	PT
21	DE	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => ignore	DE
22	DE	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax Set_Port_State.req (PRM_RPort, BLOCKED) DownTimer.stop	AC_STAT1
23	DE	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	DE
24	DE	**TopologyChangeInd(MRP_SA, t)** => MRP_LNKNReturn := MRP_LNKNRmax DownTimer.stop CLEAR_FDB(t)	DE_IDLE
25	PT_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT_IDLE
26	PT_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort != PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort , MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE

#	Current State	Event /Condition =>Action	Next State
27	PT_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_DOWN => MRP_LNKNReturn := MRP_LNKNRmax PRM_RPort := SEC_RPort SEC_RPort := RPort Set_Port_State.req (SEC_RPort, BLOCKED) DownTimer.start(MRP_LNKdownT) LinkChangeReq(PRM_RPort, MRP_LNK_DOWN, MRP_LNKNReturn * MRP_LNKdownT)	DE
28	PT_IDLE	**MAUType_Change.ind (RPort, Link_status)** /RPort == PRM_RPort && Link_status ==MRP_LNK_UP => ignore	PT_IDLE
29	PT_IDLE	**TopologyChangeInd(MRP_SA, t)** => CLEAR_FDB(t)	PT_IDLE

4.5.4.3 MRM and MRC Functions for MRP

The functions are defined in Table 110.

Table 110 – Functions

Function name	Operations
TestRingReq(t)	SETUP_TEST_RING_REQ TestTimer.start(t)
SETUP_TEST_RING_REQ	Create MRP-PDU according MRP_Test Assignments: MRP_Type := MRP_Test MRP_Prio := MRP_TS_Prio MRP_SA := MRP_TS_SA MRP_PortRole := ring port role of the used port MRP_RingState := actual ring state MRP_Type := MRP_Common MRP_SequenceID := next SequenceID MRP_DomainUUID := assigned domain UUID MRP_Type := MRP_END LMPM_N_Data.req (RPort_1, MC_TEST, SA_Port1, PRIORITY, LT, MRP-PDU) LMPM_N_Data.req (RPort_2, MC_TEST, SA_Port2, PRIORITY, LT, MRP-PDU)
TestRingInd(MRP_SA, MRP_Prio)	LMPM_N_Data.ind (S_Port,,DA, SA,,, N_SDU) MRP-PDU := N_SDU Receive MRP-PDU according MRP_Test MRP_SA := MRP_SA from MRP-PDU MRP_Prio := MRP_Prio from MRP-PDU
TopologyChangeReq(time)	SETUP_TOPOLOGY_CHANGE_REQ (MRP_TOPNRmax * time) if time == 0 CLEAR_LOCAL_FDB else TopTimer.start(MRP_TOPchgT)
SETUP_TOPOLOGY_CHANGE_REQ (t)	Create MRP-PDU according MRP_TopologyChange Assignments: MRP_Type := MRP_TopologyChange MRP_Prio := MRP_TS_Prio MRP_SA := MRP_TS_SA MRP_Interval := t MRP_Type := MRP_Common MRP_SequenceID := next SequenceID MRP_DomainUUID := assigned domain UUID MRP_Type := MRP_END LMPM_N_Data.req (RPort_1, MC_CONTROL, SA_Port1, PRIORITY, LT, MRP-PDU) LMPM_N_Data.req (RPort_2, MC_CONTROL, SA_Port2, PRIORITY, LT, MRP-PDU)

Function name	Operations
TopologyChangeInd(MRP_SA, t)	LMPM_N_Data.ind (S_Port,,DA, SA,,, N_SDU) MRP-PDU := N_SDU Receive MRP-PDU according MRP_TopologyChange MRP_SA := MRP_SA from MRP-PDU t := MRP_Interval from MRP-PDU
LinkChangeReq(RPort, LinkStatus, time)	Create MRP-PDU according MRP_LinkUp or MRP_LinkDown Assignments: if LinkStatus ==MRP_LNK_UP MRP_Type := MRP_LinkUp else MRP_Type := MRP_LinkDown MRP_SA := MRP_TS_SA MRP_PortRole := ring port role of the used port MRP_RingState := actual ring state MRP_Interval := time MRP_Type := MRP_Common MRP_SequenceID := next SequenceID MRP_DomainUUID := assigned domain UUID MRP_Type := MRP_END LMPM_N_Data.req (RPort, MC_TEST, SA_RPort, PRIORITY, LT, MRP-PDU)
LinkChangeInd(PortMode, LinkStatus)	LMPM_N_Data.ind (S_Port,,DA, SA,,, N_SDU) MRP-PDU := N_SDU Receive MRP-PDU according MRP_LinkDown or MRP_LinkUp PortMode := MRP_Blocked from MRP-PDU (shall be set to 1) if MRP_Type == MRP_LinkUp LinkStatus :=MRP_LNK_UP else LinkStatus :=MRP_LNK_DOWN
CLEAR_FDB(Time)	FDBClearTimer.start(t)
CLEAR_LOCAL_FDB	Function to Clear local FDB
INIT_FDB	Function to initialize Filtering Data Base
ADD_MAC_FDB(Destination, MAC_Address, Priroty)	Function to add MAC-Address in Filtering Data Base
SetPortState.req(RPort, State)	Function to set the port state of a ring port

4.5.4.4 FDB Clear Timer for MRP

The state table is defined in Table 111.

Table 111 – FDB Clear Timer

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> FDBClearTimer.ini	IDLE
2	IDLE	**FDBClearTimer .expired** => CLEAR_LOCAL_FDB	IDLE

4.5.4.5 Topology Change Timer for MRP

The state table is defined in Table 112.

Table 112 – Topology Change Timer

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> TopTimer.ini TC_NReturn := MRP_TOPNRmax - 1	IDLE
2	IDLE	TopTimer expired /TC_NReturn > 0 => SETUP_TOPOLOGY_CHANGE_REQ (TC_NReturn * MRP_TOPchgT) TC_NReturn-- TopTimer.start(MRP_TOPchgT)	IDLE
3	IDLE	TopTimer expired /TC_NReturn <= 0 => TC_NReturn := MRP_TOPNRmax - 1 CLEAR_LOCAL_FDB SETUP_TOPOLOGY_CHANGE_REQ (0)	IDLE

4.5.4.6 MRM Protocol Machine for MRRT activation

Figure 38 shows the principal behavior of the protocol machine for media redundancy realtime activation for MRM.

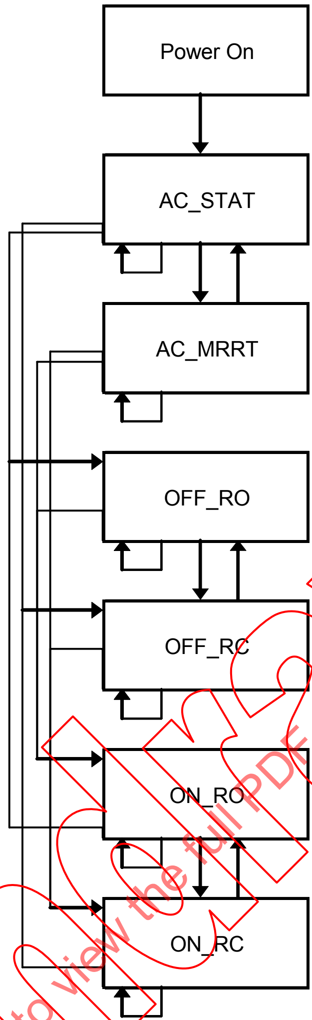

Figure 38 – MRM protocol machine

The protocol machine states perform in general the following tasks:

- **Power On**
  Initialization, the MRM shall start with media redundancy for realtime disabled, bridge forwarding mode set to cut through and frames addressed to MC_MRRT_TEST destination shall be discarded.
- **AC_STAT**
  Startup, wait for MRP ring state change indication or MRRT mode change request.
- **AC_MRRT**
  Startup after MRRT mode change request occurred.
- **OFF-RO**
  This state shall be reached if MRRT mode is DISABLED and ring is in open state.
- **OFF-RC**
  This state shall be reached if MRRT mode is DISABLED and ring is in closed state.
- **ON-RO**
  This state shall be reached if MRRT mode is ENABLED and ring is in open state.
- **ON-RC**
  This state shall be reached if MRRT mode is ENABLED and ring is in closed state. MRRT test monitoring is activated, MRRT mode at ring ports is enabled in this state only if test monitoring succeeds (all MRC in the ring with MRRT mode enabled).

Local variables of the MRM Protocol Machine for MRRT Activation protocol are listed in Table 113.

Table 113 – Local variables of MRM Protocol Machine for MRRT Activation

Name	Type	Meaning
MRRT_RingClosed	Boolean	Ring is closed (TRUE) or open (FALSE)

The MRM state Machine for MRRT Activation shall be according to Table 114.

Table 114 – MRM state machine for MRRT Activation

#	Current State	Event /Condition =>Action	Next State
1	Power On	=> MRRT_RingClosed := False Set_Bridge_State (CUT_THROUGH) ADD_MAC_FDB ({NIL}, {MC_MRRT_TEST}, ORG) Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE) MRRT_MRM_NRmax := MRRT_TSTNRmax - 1 MRRT_MRM_NReturn := 0	AC_STAT
2	AC_STAT	MRRTTestTimer expired => ignore	AC_STAT
3	AC_STAT	MRP_StateInd (RingState) /RingState == OPEN =>	OFF_RO
4	AC_STAT	MRP_StateInd (RingState) /RingState == CLOSED =>	OFF_RC
5	AC_STAT	MRRT_Set_StateInd (State) /State == ENABLE =>	AC_MRRT
6	AC_STAT	MRRT_Set_StateInd (State) /State == DISABLE => ignore	AC_STAT
7	AC_STAT	MRRT_TestRingInd (MRRT_SA) => ignore	AC_STAT
8	AC_MRRT	MRRTTestTimer expired => ignore	AC_MRRT

#	Current State	Event /Condition =>Action	Next State
9	AC_MRRT	MRP_StateInd (RingState) /RingState == OPEN => ADD_MAC_FDB ({local}, {MC_MRRT_TEST}, ORG) Set_RPort_State (RPort_1, ENABLE) Set_RPort_State (RPort_2, ENABLE)	ON_RO
10	AC_MRRT	MRP_StateInd (RingState) /RingState == CLOSED => MRRT_RingClosed := False Set_Bridge_State (STORE_AND_FORWARD) ADD_MAC_FDB ({local}, {MC_MRRT_TEST}, ORG) MRRT_TestRingReq(MRRT_TSTdefaultT)	ON_RC
11	AC_MRRT	MRRT_Set_StateInd (State) /State == DISABLE =>	AC_STAT
12	AC_MRRT	MRRT_Set_StateInd (State) /State == ENABLE => ignore	AC_MRRT
13	AC_MRRT	MRRT_TestRingInd (MRRT_SA) => ignore	AC_MRRT
14	OFF_RO	MRRTTestTimer expired => ignore	OFF_RO
15	OFF_RO	MRP_StateInd (RingState) /RingState == OPEN => ignore	OFF_RO
16	OFF_RO	MRP_StateInd (RingState) /RingState == CLOSED =>	OFF_RC
17	OFF_RO	MRRT_Set_StateInd (State) /State == DISABLE => ignore	OFF_RO
18	OFF_RO	MRRT_Set_StateInd (State) /State == ENABLE => ADD_MAC_FDB ({local}, {MC_MRRT_TEST}, ORG) Set_RPort_State (RPort_1, ENABLE) Set_RPort_State (RPort_2, ENABLE)	ON_RO

#	Current State	Event /Condition =>Action	Next State
19	OFF_RO	MRRT_TestRingInd (MRRT_SA) => ignore	OFF_RO
20	OFF_RC	MRRTTestTimer expired => ignore	OFF_RC
21	OFF_RC	MRP_StateInd (RingState) /RingState == CLOSED => ignore	OFF_RC
22	OFF_RC	MRP_StateInd (RingState) /RingState == OPEN =>	OFF_RO
23	OFF_RC	MRRT_Set_StateInd (State) /State == DISABLE => ignore	OFF_RC
24	OFF_RC	MRRT_Set_StateInd (State) /State == ENABLE => MRRT_RingClosed := False Set_Bridge_State (STORE_AND_FORWARD) ADD_MAC_FDB ({local}, {MC_MRRT_TEST}, ORG) MRRT_TestRingReq(MRRT_TSTdefaultT)	ON_RC
25	OFF_RC	MRRT_TestRingInd (MRRT_SA) => ignore	OFF_RC
26	ON_RO	MRRTTestTimer expired => ignore	ON_RO
27	ON_RO	MRP_StateInd (RingState) /RingState == OPEN => ignore	ON_RO
28	ON_RO	MRP_StateInd (RingState) /RingState == CLOSED => MRRT_RingClosed := False Set_Bridge_State (STORE_AND_FORWARD) Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE) MRRT_TestRingReq(MRRT_TSTdefaultT)	ON_RC

#	Current State	Event /Condition =>Action	Next State
29	ON_RO	MRRT_Set_StateInd (State) /State == DISABLE => Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE) REM_MAC_FDB ({local}, {MC_MRRT_TEST}, ORG)	OFF_RO
30	ON_RO	MRRT_Set_StateInd (State) /State == ENABLE => ignore	ON_RO
31	ON_RO	MRRT_TestRingInd (MRRT_SA) => ignore	ON_RO
32	ON_RC	MRRTTestTimer expired /!MRRT_RingClosed => Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE) MRRT_TestRingReq(MRRT_TSTdefaultT)	ON_RC
33	ON_RC	MRRTTestTimer expired /MRRT_RingClosed && MRRT_MRM_NReturn < MRRT_MRM_NRmax => MRRT_MRM_NReturn := MRRT_MRM_NReturn + 1 MRRT_TestRingReq(MRRT_TSTdefaultT)	ON_RC
34	ON_RC	MRRTTestTimer expired /MRRT_RingClosed && MRRT_MRM_NReturn >= MRRT_MRM_NRmax => MRRT_MRM_NReturn := 0 MRRT_RingClosed := False MRRT_TestRingReq(MRRT_TSTdefaultT)	ON_RC
35	ON_RC	MRP_StateInd (RingState) /RingState == CLOSED => ignore	ON_RC
36	ON_RC	MRP_StateInd (RingState) /RingState == OPEN => Set_Bridge_State (CUT_THROUGH) Set_RPort_State (RPort_1, ENABLE) Set_RPort_State (RPort_2, ENABLE)	ON_RO

#	Current State	Event /Condition =>Action	Next State
37	ON_RC	MRRT_Set_StateInd (State) /State == DISABLE => Set_Bridge_State (CUT_THROUGH) Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE) REM_MAC_FDB ({local}, {MC_MRRT_TEST}, ORG)	OFF_RC
38	ON_RC	MRRT_Set_StateInd (State) /State == ENABLE => ignore	ON_RC
39	ON_RC	MRRT_TestRingInd (MRRT_SA) /!MRRT_RingClosed && MRRT_SA == MRRT_TS_SA => MRRT_RingClosed := True MRRT_MRM_NReturn := 0 Set_RPort_State (RPort_1, ENABLE) Set_RPort_State (RPort_2, ENABLE)	ON_RC
40	ON_RC	MRRT_TestRingInd (MRRT_SA) /MRRT_RingClosed && MRRT_SA == MRRT_TS_SA => ignore	ON_RC
41	ON_RC	MRRT_TestRingInd (MRRT_SA) /MRRT_SA != MRRT_TS_SA => ignore	ON_RC

4.5.4.7 MRC Protocol Machine for MRRT activation

Figure 39 shows the principal behavior of the protocol machine for media redundancy realtime activation for MRC.

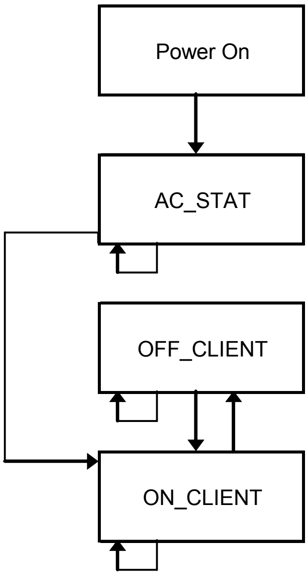

Figure 39 – MRC protocol machine for MRRT

The protocol machine states perform in general the following tasks:

- **Power On**
  Initialization, the MRC shall start with media redundancy for realtime disabled, bridge forwarding mode set to cut through and frames addressed to MC_MRRT_TEST destination shall be discarded.
- **AC_STAT**
  Startup, wait for MRRT mode ENABLE request.
- **ON-CLIENT**
  This state shall be reached if MRRT mode is ENABLED. Frames addressed to MC_MRRT_TEST destination shall be forwarded between ring ports.
- **OFF-CLIENT**
  This state shall be reached if MRRT mode is DISABLED. Frames addressed to MC_MRRT_TEST destination shall be discarded.

The MRC state Machine for MRRT Activation is shown in Table 115.

Table 115 – MRC state machine for MRRT Activation

#	Current State	Event /Condition =>Action	Next State
1	Power On	 => Set_Bridge_State (CUT_THROUGH) ADD_MAC_FDB ({NIL}, {MC_MRRT_TEST}, ORG) Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE)	AC_STAT
2	AC_STAT	**MRRT_Set_StateInd (State)** /State == ENABLE => Set_RPort_State (RPort_1, ENABLE) Set_RPort_State (RPort_2, ENABLE) ADD_MAC_FDB({RPort_1, RPort_2}, {MC_MRRT_TEST}, ORG)	ON_CLIENT
3	AC_STAT	**MRRT_Set_StateInd (State)** /State == DISABLE => ignore	AC_STAT
4	OFF_CLIENT	**MRRT_Set_StateInd (State)** /State == DISABLE => ignore	OFF_CLIEN T
5	OFF_CLIENT	**MRRT_Set_StateInd (State)** /State == ENABLE => Set_RPort_State (RPort_1, ENABLE) Set_RPort_State (RPort_2, ENABLE) ADD_MAC_FDB({RPort_1, RPort_2}, {MC_MRRT_TEST}, ORG)	ON_CLIENT
6	ON_CLIENT	**MRRT_Set_StateInd (State)** /State == DISABLE => REM_MAC_FDB({RPort_1, RPort_2}, {MC_MRRT_TEST}, ORG) Set_RPort_State (RPort_1, DISABLE) Set_RPort_State (RPort_2, DISABLE)	OFF_CLIEN T
7	ON_CLIENT	**MRRT_Set_StateInd (State)** /State == ENABLE => ignore	ON_CLIENT

4.5.4.8 MRM and MRC functions for MRRT activation

The functions are defined in Table 116.

Table 116 – MRM and MRC functions

Function name	Operations
MRRT_TestRingReq(t)	MRRT_SETUP_TEST_RING_REQ MRRTTestTimer.start(t)
MRRT_SETUP_TEST_RING_REQ	Create MRRT-PDU according MRRT_Test Assignments: MRRT_Type := MRRT_Test MRRT_SA := MRRT_TS_SA (same as MRP_TS_SA) MRRT_Type := MRRT_Common MRRT_SequenceID := next SequenceID MRRT_DomainUUID := MRP_DomainUUID of ring ports MRRT_Type := MRRT_END LMPM_A_Data.req (RPort_1, MC_MRRT_TEST, SA_Port1, PRIORITY, LT, MRRT-PDU) LMPM_A_Data.req (RPort_2, MC_MRRT_TEST, SA_Port2, PRIORITY, LT, MRRT-PDU)

MRRT_TestRingInd(MRRT_SA)	LMPM_A_Data.ind (S_Port,,DA, SA,,, N_SDU)
	MRRT-PDU := N_SDU
	
	Receive MRRT-PDU according MRRT_Test
	if S_Port is ring port
	{
	MRRT_SA := MRRT_SA from MRRT-PDU
	}
	else discard MRRT-PDU

ADD_MAC_FDB(Destination, MAC-Address, Priority)	Function to add MAC-Address in Forwarding Data Base
REM_MAC_FDB(Destination, MAC-Address, Priority)	Function to remove MAC-Address from Forwarding Data Base
STORE_AND_FORWARD	Function to set bridge forwarding mode to store&forward
CUT_THROUGH	Function to set bridge forwarding mode to cut through
Set_RPort_State(Port, MRRT_Mode)	Function to enable or disable MRRT mode at port
RPort_1, RPort_2	Ring ports (equal to MRP ring ports)
MRRT_Set_StateInd (State)	MRRT mode change request (ENABLE or DISABLE)
MRP_StateInd (RingState)	MRP ring state change indication (MRP_RingState change from ring open to ring closed)

4.6 Real-time cyclic

4.6.1 FAL syntax description

4.6.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.6.1.2 RTC APDU abstract syntax

Table 117 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 117 shall represent the content of the DLSDU in Table 4.

Table 117 – RTC APDU syntax

APDU name	APDU structure
RTC-PDU	C_SDU ^ CBA_SDU, APDU_Status
UDP-RTC-PDU	IPHeader, UDPHeader, FrameID, RTC-PDU

Table 118 defines structures for substitutions of elements of the APDU structures shown in Table 117.

Table 118 – RTC substitutions

Substitution name	Structure
C_SDU	{[DataItem], [GAP*]} *, [RTCPadding*] [a] [b] The maximum C_SDU size shall be not larger than 1 440 octets.
DataItem	{[IOCS*], [DataObjectElement*]} *
SubstituteDataItem	{[IOCS*] [c], [SubstituteDataObjectElement*]} *
DataObjectElement	[Data*], IOPS
SubstituteDataObjectElement	[Data*], SubstituteDataValid
CBA_SDU	CBAHeader, CBADataSet*,[RTCPadding*] [a] [b]
CBAHeader	CBAVersion, CBAFlags, CBACount
CBADataSet	CBALength, CBAQualityCode, CBAValue
APDU_Status	CycleCounter, DataStatus, TransferStatus

[a] In case of RTC-PDU the number of padding octets shall be in the range of 0...40 depending on the DataItem size. In case of UDP-RTC-PDU the number of padding octets shall be in the range of 0...12 depending on the DataItem size.

[b] In case of RT_CLASS_3 transportation the number of padding octets is given by the engineering system.

[c] The value shall contain the same value as the field DataItem.IOCS.

4.6.2 FAL transfer syntax

4.6.2.1 Coding section of Data-RTC-PDU specific fields

4.6.2.1.1 Overview

The field Data and IOxS shall also be used to encode user data for all other PDU types.

4.6.2.1.2 Coding of the field CycleCounter

This field shall be coded as data type Unsigned16. One increment represents a time value of 31,25 µs for RTC-PDU and 1 ms for UDP-RTC-PDU. Each RTC-PDU contains its local CycleCounter. The receiver of the RTC-PDU shall use this field to detect repetitions, loss of frames, and timeliness of data.

Table 119 defines the action of the consumer according to the difference between the CycleCounter of the last received RTC-PDU or UDP-RTC-PDU and the value of the current received PDU.

Table 119 – CycleCounter Difference

Value (hexadecimal)	Meaning	Use
0x0000 – 0x0FFF	The consumer votes the received frame as older than the stored frame	Frame is dropped
0x1000 – 0xFFFF	The consumer votes the received frame as newer than the stored frame	Frame is processed

The CycleCounter defines a time window between the old and the new frame of 1,92 s to vote as new for RTC-PDUs and 61,44 s to vote as new for UDP-RTC-PDUs.

NOTE If necessary, the local CycleCounter may tightly synchronized global in the network.

A device shall maintain a local 64 bit counter with an increment of 31,25 us. The bits 0 to 15 shall be used for CycleCounter in RTC-PDUs. The bits 5 to 20 shall be used for CycleCounter in UDP-RTC-PDUs as shown in Figure 40.

The difference shall be calculated with 16 bit arithmetics.

IF (CycleCounter(NEW) >= CycleCounter(STORED)) (46)

Difference = CycleCounter(NEW) - CycleCounter(STORED)

ELSE

Difference = 0xFFFF - CycleCounter(STORED) + CycleCounter(NEW)

If the difference value == 0 then the CycleCounter(NEW) is older.

If the difference value > 0xF000 then the CycleCounter(NEW) is older.

If the difference value <= 0xF000 then the CycleCounter(NEW) is newer.

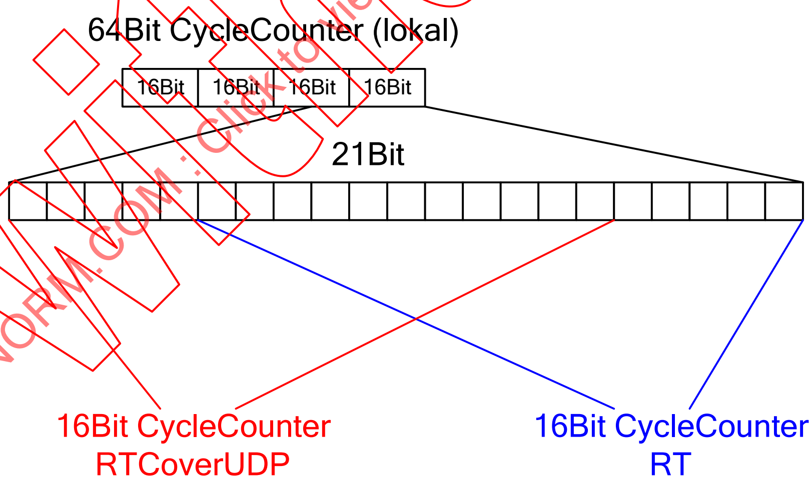

Figure 40 – Structure of the CycleCounter

4.6.2.1.3 Coding of the field DataStatus

The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0: DataStatus.State

This field shall be coded with the values according to Table 120.

Table 120 – DataStatus.State

Value (hexadecimal)	Meaning
0x00	IOCR state is backup
0x01	IOCR state is primary

Bit 1: DataStatus.reserved_1
This field shall be set according to 3.6.3.2.

Bit 2: DataStatus.DataValid
This field shall be coded with the values according to Table 121.

Table 121 – DataStatus.DataValid

Value (hexadecimal)	Meaning
0x00	DataItem invalid
0x01	DataItem valid

Bit 3: DataStatus.reserved_2
This field shall be set to zero but not checked by the receiver.

Bit 4: DataStatus.ProviderState
This field shall be coded with the values according to Table 122.

Table 122 – DataStatus.ProviderState

Value (hexadecimal)	Meaning
0x00	Stop
0x01	Run

Bit 5: DataStatus.StationProblemIndicator
This field shall be coded with the values according to Table 123.

Table 123 – DataStatus.StationProblemIndicator

Value (hexadecimal)	Meaning
0x00	Problem detected
0x01	Normal Operation

Bit 6: DataStatus.reserved_3
This field shall be set to zero but not checked by the receiver.

Bit 7: DataStatus.reserved_4
This field shall be set to zero but not checked by the receiver.

4.6.2.1.4 Coding of the field TransferStatus

This field shall be coded as data type Unsigned8. This field shall be set to zero for RT_CLASS_UDP, RT_CLASS_1, and RT_CLASS_2 frames. For RT_CLASS_3 the value shall be set according to Table 124.

Table 124 – TransferStatus for RT_CLASS_3

Bit position	Name	Value	Meaning
0	AlignmentOrFrameChecksumError	0	No frame checksum error or alignment error
		1	A frame checksum error or alignment error and no MAC receive buffer overflow has appeared
1	WrongLengthError	0	No length error
		1	A length error has appeared
2	MACReceiveBufferOverflow	0	No MAC receive buffer overflow
		1	A MAC receive buffer overflow has appeared
3	RT_CLASS_3 Error	0	No RT_CLASS_3 error
		1	RT_CLASS_3 error E.g. the frame has not been received in time or has not had the expected frame type or has not had the expected frame ID or has not had the expected Source MAC address (this is optionally checked) or in case of an underrun, e.g. the bridge received a header, but the data have not arrived on time
Other	None	Reserved	Reserved

4.6.2.1.5 Coding section related to decentralized periphery

4.6.2.1.5.1 Coding of the field Data

One of the following data types shall be used for each Data field:

- Boolean
- Integer
- Unsigned
- Floating Point
- Visible String
- OctetString
- Binary Date
- Time of Day
- Time-Difference
- Network Time
- Network Time Difference

4.6.2.1.5.2 Coding section related to IOxS (IOPS, IOCS, SubstituteDataValid)

4.6.2.1.5.2.1 Coding of the field IOPS

The coding of these fields shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0: IOxS.Extension
This field shall be coded with the values according to Table 125.

Table 125 – IOxS.Extension

Value (hexadecimal)	Meaning
0x00	No IOxS octet follows
0x01	One more IOxS octet follows

Bit 1 to 4: IOxS.reserved
This field shall be set according to 3.6.3.2.

Bit 6 to 5: IOxS.Instance
This field shall be coded with the values according to Table 126 if the IOxS.DataState bit is set to bad. If this bit is set to good the IOxS instance bits are don't care and shall be set to zero.

Table 126 – IOCS.Instance

Value (hexadecimal)	Meaning
0x00	Detected by subslot
0x01	Detected by slot
0x02	Detected by IO device
0x03	Detected by IO controller

Bit 7: IOxS.DataState
This field shall be coded with the values according to Table 127.

Table 127 – IOxS.DataState

Value (hexadecimal)	Meaning
0x00	Bad Data field may not contain valid user data – the receiver shall use its pre-configured values (zero, last valid value, or default value) instead of transmitted data field values SubstituteDataValid = FALSE
0x01	Good Data field shall contain valid user data SubstituteDataValid = TRUE

4.6.2.1.5.2.2 Coding of the field IOCS

The coding of these fields shall be according to 4.6.2.1.5.2.1.

4.6.2.1.5.2.3 Coding of the field SubstituteDataValid

The coding of these fields shall be according to 4.6.2.1.5.2.1.

4.6.2.1.6 Coding section related to distributed automation

4.6.2.1.6.1 Coding of the field CBAValue

One of the following data types shall be used for each CBAValue field:

- VARIANT_BOOL
- char
- unsigned char
- short
- unsigned short
- long
- unsigned long
- float
- double
- date
- BSTR
- SAFEARREY
- struct

4.6.2.1.6.2 Coding of the field CBAVersion

This field shall be coded as data type Unsigned8. This field shall be set to 0x11.

4.6.2.1.6.3 Coding of the field CBAFlags

This field shall be coded as data type Unsigned8. This field shall be set to 0x00.

4.6.2.1.6.4 Coding of the field CBACount

This field shall be coded as data type Unsigned16. This field shall be set to the number of following CBADataSet fields.

4.6.2.1.6.5 Coding of the field CBALength

This field shall be coded as data type Unsigned16. This field shall be set to the number of octets of the fields CBALength, CBAQualityCode, and CBAValue.

4.6.2.1.6.6 Coding of the field CBAQualityCode

This field shall be coded as data type ITEMQUALITYDEF.

4.6.3 Application Relationship Protocol Machines (ARPMs)

4.6.3.1 PPM

4.6.3.1.1 Primitive definitions

4.6.3.1.1.1 Primitives exchanged between PPM and PPM user

The service primitives including their associated parameters issued by PPM user received by PPM and vice versa are described in the RTC ASE in the service definition.

4.6.3.1.1.2 Primitives exchanged between PPM and LMPM

The service primitives including their associated parameters issued by PPM received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.6.3.1.1.3 Parameters of PPM primitives

The parameters used with the primitives exchanged between the PPM user and the PPM are described in FAL Service Definition.

4.6.3.1.2 State machine description

The W-START state indicates that an initialisation is needed. The Activate service sets the machine in the RUN state waiting for an Provider Data service requests and Time events. After issuing the transmission of data the CRUN state is used to wait for a confirmation. Receiving the confirmation will bring back the machine to the RUN state. An error or a Close service request is needed to re-enter the W-START state. A Close in CRUN has to wait for the outstanding confirmation in the WCON state before entering W-START.

Local variables of the PPM

Buffer_Data (OctetString)
This local variable contains the data (encoded as DataItem*) which shall be used as storage of the data dedicated for conveyance of the next Data-RTC-PDU or CL-RPC-PDU with NDRDataCyclic.

NOTE DataItem* includes all DataItems for the dedicated PDU.

Buffer_Status (Unsigned 16)
This local variable contains the status (encoded as APDU_STatus) which shall be used as storage of the status dedicated for conveyance of the next Data-RTC-PDU or CL-RPC-PDU with NDRDataCyclic.

New_Stat (Boolean)
This local variable will be used to indicate the first successful transmission of data after Open.

4.6.3.1.3 PPM state table

Table 128 contains the complete description of the PPM state table.

Table 128 – PPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	**PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status)** => New_Stat := FALSE; Buffer_Data := Default_Values Buffer_Status := Default_Status LMPM_Schedule_add.req (CREP, ReductionRatio, Phase, Sequence) PPM_Activate.cnf (+)(CREP)	RUN
2	W-START	**PPM_Close.req(CREP)** => PPM_Close.cnf (+)(CREP)	W-START
3	W-START	**PPM_Set_Prov_Data.req (CREP,Data)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
4	W-START	**PPM_Set_Prov_Status.req (CREP,D_Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
5	W-START	**C_Data_cnf** => ignore	W-START
6	RUN	**PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	RUN

#	Current State	Event /Condition =>Action	Next State
7	RUN	**PPM_Close.req(CREP)** => LMPM_Schedule_remove.req(CREP) PPM_Close.cnf (+)(CREP)	W-START
8	RUN	**PPM_Set_Prov_Data.req (CREP,Data)** => Buffer_Data := Data PPM_Set_Prov_Data.cnf(+) (CREP)	RUN
9	RUN	**PPM_Set_Prov_Status.req (CREP,D_Status)** => Buffer_Status := D_Status PPM_Set_Prov_Status.cnf(+) (CREP)	RUN
10	RUN	**C_Data_cnf** => ignore	RUN
11	RUN	**LMPM_Time_Event.ind(CREP, PortList)** /PortList == NIL => C_SDU := Buffer_Data, APDU_Status := (Cycle_Count, Buffer_Status) C_Data_req(PortList)	CRUN
12	RUN	**LMPM_Time_Event.ind(CREP, PortList)** /PortList !=NIL => C_SDU := Buffer_Data, APDU_Status := (Cycle_Count, Buffer_Status) C_Data_req(PortList)	CRUN
13	CRUN	**PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	CRUN
14	CRUN	**PPM_Close.req(CREP)** => LMPM_Schedule_remove.req(CREP) PPM_Close.cnf (+)(CREP)	WCON
15	CRUN	**C_Data_cnf** /LMPM_status != LS/IV && !New_Stat => New_Stat := TRUE PPM_Start.ind(CREP)	RUN
16	CRUN	**C_Data_cnf** /LMPM_status != LS/IV && New_Stat =>	RUN
17	CRUN	**C_Data_cnf** /LMPM_status == LS/IV => PPM_Error.ind(CREP,No_Data_Send)	RUN

#	Current State	Event /Condition =>Action	Next State
18	CRUN	**LMPM_Time_Event.ind(CREP, PortList)** => PPM_Error.ind(CREP,No_Data_Send)	CRUN
19	CRUN	**PPM_Set_Prov_Data.req (CREP,Data)** => Buffer_Data := Data PPM_Set_Prov_Data.cnf(+) (CREP)	CRUN
20	CRUN	**PPM_Set_Prov_Status.req (CREP,D_Status)** => Buffer_Status := D_Status PPM_Set_Prov_Status.cnf(+) (CREP)	CRUN
21	WCON	**PPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, TxOption, ReductionRatio, Phase, Sequence, Default_Values, Default_Status)** => New_Stat := FALSE; Buffer_Data := Default_Values Buffer_Status := Default_Status LMPM_Schedule_add.req (CREP, ReductionRatio, Phase, Sequence) PPM_Activate.cnf (+)(CREP)	CRUN
22	WCON	**PPM_Close.req(CREP)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Close.cnf (-)(CREP,ERRCLS,ERRCODE)	WCON
23	WCON	**C_Data_cnf** /LMPM_status != LS/IV =>	W-START
24	WCON	**C_Data_cnf** /LMPM_status == LS/IV => PPM_Error.ind(CREP,No_Data_Send)	W-START
25	WCON	**LMPM_Time_Event.ind(CREP, PortList)** => PPM_Error.ind(CREP,No_Data_Send)	WCON
26	WCON	**PPM_Set_Prov_Data.req (CREP,Data)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	WCON
27	WCON	**PPM_Set_Prov_Status.req (CREP,D_Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE PPM_Set_Prov_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	WCON

4.6.3.1.4 Functions

Table 129 contains the functions or macros used by the PPM, their arguments and their descriptions.

Table 129 – Functions used by the PPM

Function name	Description
C_Data_cnf	if (txOption == RTC) LMPM_C_Data.cnf (CREP, LMPM_status, Cycle) if (txOption == UDP) UDP_C_Data.cnf (IPPort, CREP, LMPM_status, Cycle)
C_Data_req(PortList)	if (txOption == RTC) LMPM_C_Data.req (CREP, PortList, DA, SA, Prio, C_SDU, APDU_Status) if (txOption == UDP) UDP_C_Data.req (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)

4.6.3.2 CPM

4.6.3.2.1 Primitive definitions

4.6.3.2.1.1 Primitives exchanged between CPM and CPM user

The service primitives including their associated parameters issued by CPM user received by CPM and vice versa are described in the RTC ASE in the service definition.

4.6.3.2.1.2 Primitives exchanged between CPM and LMPM

The service primitives including their associated parameters issued by CPM received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.6.3.2.1.3 Parameters of CPM primitives

The parameters used with the primitives exchanged between the CPM user and the CPM are described in FAL Service Definition.

4.6.3.2.2 State machine description

The W-START state indicates that the initialisation is needed. The Activate service sets the machine in the RUN state or in the FRUN state. The state machine is waiting for the first LMPM_C_Data service indication in these states. After passing the Start indication to the user the RUN state is entered. A timeout will force a state transition back to FRUN (combined with a Stop indication). A Close service request is needed to reenter the W-START state.

Each time receiving valid C_Data.ind the time is read and stored in a local variable together with a Flag.

Local variables of the CPM

Cycle (Unsigned 16)
This local variable contains the last valid Cycle Counter conveyed with a LMPM_C_Data indication with valid data.

RecvCnt (Unsigned 16)
This local variable contains the counter value of the received cyclic PDU since last Cons Status request.

Buffer_Data (Octed String)
This local variable contains the last valid service data unit conveyed with a LMPM_C_Data indication.

Buffer_Status (Unsigned 16)
This local variable contains the last valid APDU_Status conveyed with a LMPM_C_Data indication.

New_Data (Boolean)
This local variable signals the receipt of a valid LMPM_C_Data indication (set to FALSE by a Get_Cons_Data request).

WDt (Unsigned 16)
This local variable contains the counter value of the time events since last LMPM_C_Data indication.

DHt (Unsigned 16)
This local variable contains the counter value of the time events since last LMPM_C_Data indication with valid data.

4.6.3.2.3 CPM state table

Table 130 contains the complete description of the CPM state table.

Table 130 – CPM state table

#	Current State	Event /Condition =>Action	Next State
1	W-START	**CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status)** /StartTon = TRUE && RTClass != RTClass3 => InitCy := TRUE MEDIA_RED_LOST := FALSE Store DA,SA,FrameID,Prio, VLAN, Timeout_Base, WatchDogFactor, DataHoldFactor, Default_Value, Default_Status New_Data := FALSE, for all ports of interface do: WDt (Port):= 0, DHt(Port) := 0 Cycle := 0, RecvCnt := 0, Buffer_Data := Default_Value, Buffer_Status := Default_Status Primary := TRUE LMPM_Schedule_add.req (CREP, ReductionRatio:=Timeout_Base, Phase:=0, Sequence:=0) CPM_Activate.cnf (+)(CREP)	RUN
2	W-START	**CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status)** /StartTon = FALSE && RTClass != RTClass3 => MEDIA_RED_LOST := FALSE Store DA,SA,FrameID,Prio, VLAN, Timeout_Base, WatchDogFactor, DataHoldFactor, Default_Value, Default Status New_Data := FALSE, for all ports of interface do: WDt (Port):= 0, DHt(Port) := 0 Cycle := 0, RecvCnt := 0, Buffer_Data := Default_Value, Buffer_Status := Default_Status Primary := TRUE LMPM_Schedule_add.req (CREP, ReductionRatio:=Timeout_Base, Phase:=0, Sequence:=0) CPM_Activate.cnf (+)(CREP)	FRUN

#	Current State	Event /Condition =>Action	Next State
3	W-START	**CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status)** /StartTon = TRUE && RTClass == RTClass3 => InitCy := TRUE MEDIA_RED_LOST := FALSE Store DA,SA,FrameID,Prio, VLAN, Timeout_Base, WatchDogFactor, DataHoldFactor, Default_Value, Default_Status New_Data := FALSE, for all ports of interface do: WDt (Port):= 0, DHt(Port) := 0 Cycle := 0, RecvCnt := 0, Buffer_Data := Default_Value, Buffer_Status := Default_Status Primary := TRUE CPM_Activate.cnf (+)(CREP)	RUN
4	W-START	**CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Length, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status)** /StartTon = FALSE && RTClass == RTClass3 => MEDIA_RED_LOST := FALSE Store DA,SA,FrameID,Prio, VLAN, Timeout_Base, WatchDogFactor, DataHoldFactor, Default_Value, Default Status New_Data := FALSE, for all ports of interface do: WDt (Port):= 0, DHt(Port) := 0 Cycle := 0, RecvCnt := 0, Buffer_Data := Default_Value, Buffer_Status := Default_Status Primary := TRUE CPM_Activate.cnf (+)(CREP)	FRUN
5	W-START	**CPM_Close.req(CREP)** => CPM_Close.cnf (+)(CREP)	W-START
6	W-START	**CPM_Get_Cons_Status.req (CREP)** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Get_Cons_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START
7	W-START	**CPM_Set_RedRole.req (CREP, RedRole)** => Primary := RedRole CPM_Set_RedRole.cnf (CREP)	W-START
8	W-START	**CPM_Get_Cons_Data.req (CREP)** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Get_Cons_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	W-START

#	Current State	Event /Condition =>Action	Next State
9	W-START	**C_Data_ind** => ignore	W-START
10	FRUN	**CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Lenth, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	FRUN
11	FRUN	**CPM_Close.req(CREP)** /RTClass != RTClass3 => LMPM_Schedule_remove.req(CREP) CPM_Close.cnf (+)(CREP)	W-START
12	FRUN	**CPM_Close.req(CREP)** /RTClass == RTClass3 => CPM_Close.cnf (+)(CREP)	W-START
13	FRUN	**CPM_Get_Cons_Status.req (CREP)** => Status := Cycle, Buffer_Status, RecvCounter := RecvCnt, RecvCnt := 0 CPM_Get_Cons_Status.cnf(+) (CREP, Status, RecvCounter)	FRUN
14	FRUN	**CPM_Set_RedRole.req (CREP, RedRole)** => Primary := RedRole CPM_Set_RedRole.cnf (CREP)	FRUN
15	FRUN	**CPM_Get_Cons_Data.req (CREP)** => Data := Buffer_Data, New_Flag := New_Data, New_Data := FALSE CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)	FRUN
16	FRUN	**C_Data_ind** /MEDIA_RED_LOST == FALSE && APDU_Status.DataStatus.DataValid && CHECK_PAR_C => InitCy := FALSE, New_Data := TRUE, DHt(Port) := 0, WDt(Port) := 0, Cycle:= APDU_Status.Cycle_Counter, Buffer_Data := Filter (Data) Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1 CPM_New_Cons_Data.ind(CREP, APDU_Status)	RUN

#	Current State	Event /Condition =>Action	Next State
17	FRUN	**C_Data_ind** /MEDIA_RED_LOST == TRUE && APDU_Status.DataStatus.DataValid && CHECK_PAR_C => InitCy := FALSE, MEDIA_RED_LOST := FALSE New_Data := TRUE, DHt(Port) := 0, WDt(Port) := 0, Cycle:= APDU_Status.Cycle_Counter, Buffer_Data := Filter (Data) Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1 CPM_New_Cons_Data.ind(CREP, APDU_Status)	RUN
18	FRUN	**C_Data_ind** /!APDU_Status.DataStatus.DataValid && !Primary && CHECK_PAR_C => DHt(Port) := 0, WDt(Port) := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1	FRUN
19	FRUN	**C_Data_ind** /!APDU_Status.DataStatus.DataValid && Primary && CHECK_PAR_C => WDt(Port) := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1	FRUN
20	FRUN	**C_Data_ind** /!CHECK_PAR_C => ignore	FRUN
21	FRUN	**LMPM_Time_Event.ind(CREP)** => ignore	FRUN
22	RUN	**CPM_Activate.req(CREP, DA, SA, FrameID, Prio, VLAN, Exp_Lenth, StartTon, Timeout_Base, WatchdogFactor, DataHoldFactor, Default_Value, Default_Status)** => ERRCLS := CTXT ERRCODE := INVALID_STATE CPM_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	RUN
23	RUN	**CPM_Close.req(CREP)** /RTClass != RTClass3 => LMPM_Schedule_remove.req(CREP) CPM_Close.cnf (+)(CREP)	W-START
24	RUN	**CPM_Close.req(CREP)** /RTClass == RTClass3 => CPM_Close.cnf (+)(CREP)	W-START

<table>
<tr><th>#</th><th>Current State</th><th>Event
/Condition
=>Action</th><th>Next State</th></tr>
<tr><td>25</td><td>RUN</td><td>CPM_Get_Cons_Status.req (CREP)
=>
Status := Cycle, Buffer_Status,
RecvCounter := RecvCnt,
RecvCnt := 0
CPM_Get_Cons_Status.cnf(+) (CREP, Status, RecvCounter)</td><td>RUN</td></tr>
<tr><td>26</td><td>RUN</td><td>CPM_Set_RedRole.req (CREP, RedRole)
=>
Primary := RedRole
CPM_Set_RedRole.cnf (CREP)</td><td>RUN</td></tr>
<tr><td>27</td><td>RUN</td><td>CPM_Get_Cons_Data.req (CREP)
=>
Data := Buffer_Data,
New_Flag := New_Data,
New_Data := FALSE
CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)</td><td>RUN</td></tr>
<tr><td>28</td><td>RUN</td><td>C_Data_ind
/MEDIA_RED_LOST == FALSE &&
((APDU_Status.CycleCounter-Cycle)>> || InitCy) &&
(APDU_Status.DataStatus.DataValid) && CHECK_PAR_C
=>
InitCy := FALSE,
New_Data := TRUE,
DHt(Port) := 0, WDt(Port) := 0,
Cycle:= APDU_Status.Cycle_Counter,
Buffer_Data := Filter (Data)
Buffer_Status := APDU_Status
RecvCnt := RecvCnt + 1
CPM_New_Cons_Data.ind(CREP, APDU_Status)</td><td>RUN</td></tr>
<tr><td>29</td><td>RUN</td><td>C_Data_ind
/MEDIA_RED_LOST == TRUE
((APDU_Status.CycleCounter-Cycle)>> || InitCy) &&
(APDU_Status.DataStatus.DataValid) && CHECK_PAR_C
=>
InitCy := FALSE,
MEDIA_RED_LOST := FALSE
New_Data := TRUE,
DHt(Port) := 0, WDt(Port) := 0,
Cycle:= APDU_Status.Cycle_Counter,
Buffer_Data := Filter (Data)
Buffer_Status := APDU_Status
RecvCnt := RecvCnt + 1
CPM_New_Cons_Data.ind(CREP, APDU_Status)</td><td>RUN</td></tr>
<tr><td>30</td><td>RUN</td><td>C_Data_ind
/((APDU_Status.CycleCounter-Cycle)>> || InitCy) &&
!APDU_Status.DataStatus.DataValid && !Primary && CHECK_PAR_C
=>
InitCy := FALSE,
DHt(Port) := 0, WDt(Port) := 0,
Buffer_Status := APDU_Status
RecvCnt := RecvCnt + 1</td><td>RUN</td></tr>
</table>

#	Current State	Event /Condition =>Action	Next State
31	RUN	**C_Data_ind** /((APDU_Status.CycleCounter-Cycle)<< && InitCy) && (!APDU_Status.DataStatus.DataValid && Primary)) && CHECK_PAR_C => InitCy := FALSE, WDt(Port) := 0, Buffer_Status := APDU_Status RecvCnt := RecvCnt + 1	RUN
32	RUN	**C_Data_ind** /((APDU_Status.CycleCounter-Cycle)<< && !InitCy) || !CHECK_PAR_C => ignore	RUN
33	RUN	**LMPM_Time_Event.ind(CREP, PortList)** /for every Port in PortList: DHt(Port) > (DataHoldTime-1) && WDt(Port) > (WatchDogTime-1) => for every Port in PortList: DHt(Port) := 0 WDt(Port) := 0 CPM_NoData.ind(CREP) CPM_Stop.ind(CREP)	FRUN
34	RUN	**LMPM_Time_Event.ind(CREP, PortList)** /for every Port in PortList: DHt(Port) > (DataHoldTime-1) && for at least one Port in PortList: WDt(Port) < (WatchDogTime) => for every Port in PortList: DHt(Port) := 0 WDt(Port) := WDt(Port)+1 CPM_Stop.ind(CREP)	FRUN
35	RUN	**LMPM_Time_Event.ind(CREP, PortList)** /for every Port in PortList: WDt(Port) > (WatchDogTime-1) && for at least one Port in PortList: DHt(Port) < (DataHoldTime) && PortList != NIL => for every Port in PortList: DHt(Port) := DHt(Port) +1 WDt(Port) := 0 MEDIA_RED_LOST := TRUE CPM_NoData.ind(CREP)	RUN

#	Current State	Event /Condition =>Action	Next State
36	RUN	**LMPM_Time_Event.ind(CREP, PortList)** /for every Port in PortList: DHt(Port) < (DataHoldTime) && WDt(Port) < (WatchDogTime) => for every Port in PortList: DHt(Port) := DHt(Port) +1 WDt(Port) := WDt(Port) +1	RUN

4.6.3.2.4 Functions

Table 131 contains the functions or macros used by the CPM, their arguments and their descriptions.

Table 131 – Functions used by the CPM

Function name	Description
C_Data_ind	if (txOption == RTC) LMPM_C_Data.ind (CREP, Port, DA, SA, Prio, C_SDU, APDU_Status) if (txOption == UDP) UDP_C_Data.ind (IPPort, CREP, DA, SA, Prio, C_SDU, APDU_Status)
<<	Result of the subtraction of two 16 bit unsigned integers is between -0xfff and 0 Sign overflows will not be calculated
>>	Means not << (result between 0x8000 and 0x1000 or between 1 and 0x7FFF)
CHECK_PAR_C	C_SDU.Length == Exp_Length && SA == stored SA && APDU_Status.TransferStatus == 0 && if (txOption == UDP) IPPort == Exp_IPPort

4.7 Real-time acyclic

4.7.1 RTA syntax description

4.7.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.7.1.2 RTA APDU abstract syntax

Table 132 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 132 shall represent the content of the DLSDU in Table 4.

Table 132 – RTA APDU syntax

APDU name	APDU structure
RTA-PDU	DATA-RTA-PDU ^ ACK-RTA-PDU ^ NACK-RTA-PDU ^ ERR-RTA-PDU
UDP-RTA-PDU	IPHeader, UDPHeader, FrameID, RTA-PDU

Table 133 defines structures for substitutions of elements of the APDU structures shown in Table 132.

Table 133 – RTA substitutions

Substitution name	Structure
Reference	AlarmDstEndpoint, AlarmSrcEndpoint
FlagsSequence	AddFlags, SendSeqNum, AckSeqNum
DATA-RTA-PDU	Reference, PDUType (=1), FlagsSequence, VarPartLen (1 – 1 432) [a], RTA-SDU, RTAPadding* [b]
ACK-RTA-PDU	Reference, PDUType (=3), FlagsSequence, VarPartLen (=0), RTAPadding* [c]
NACK-RTA-PDU	Reference, PDUType (=2), FlagsSequence, VarPartLen (=0), RTAPadding* [d]
ERR-RTA-PDU	Reference, PDUType (=4), FlagsSequence, VarPartLen (=4), PNIOStatus, RTAPadding* [e]

[a] The maximum VarPartLen is calculated to fit in the RTA-PDU and in the UDP-RTA-PDU.

[b] The number of padding octets shall be in the range of 0..31 depending on the RTA-SDU size.

[c] The number of padding octets shall be 32 for RTA_CLASS_1 and shall be 4 for RTA_CLASS_UDP

[d] The number of padding octets shall be 32 for RTA_CLASS_1 and shall be 4 for RTA_CLASS_UDP.

[e] The number of padding octets shall be 28 for RTA_CLASS_1 and shall be 0 for RTA_CLASS_UDP.

4.7.2 RTA transfer syntax

4.7.2.1 Coding section related to RTA-PDUs specific fields

4.7.2.1.1 Coding of the field AlarmDstEndpoint

This field shall be coded as data type Unsigned16.

4.7.2.1.2 Coding of the field AlarmSrcEndpoint

This field shall be coded as data type Unsigned16.

NOTE The value of the field reference for destination is provided by the receiver. The value of the field reference for the source is provided by the sender. The fields are specific for a connection and exchanged during context management operation.

4.7.2.1.3 Coding of the field PDUType

The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0 – 3: PDUType.Type
This field shall be coded with the values according to Table 134.

Table 134 – PDUType.Type

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	
0x01	RTA_TYPE_DATA	Shall only be used to encode the DATA-RTA-PDU
0x02	RTA_TYPE_NACK	Shall only be used to encode the NACK-RTA-PDU
0x03	RTA_TYPE_ACK	Shall only be used to encode the ACK-RTA-PDU
0x04	RTA_TYPE_ERR	Shall only be used to encode the ERR-RTA-PDU
0x05 – 0x0F	Reserved	

Bit 4 – 7: PDUType.Version
This field shall be coded with the values according to Table 135.

Table 135 – PDUType.Version

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Version 1 of the protocol
0x02 – 0x0F	Reserved

4.7.2.1.4 Coding of the field AddFlags

The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0 – 3: AddFlags.WindowSize
This field shall be set to one.

Bit 4: AddFlags.TACK
This field shall be set by the sender within a RTA-PDU to control the acknowledge behavior of the receiver.

The value one shall be set for immediate acknowledge in a DATA-RTA-PDU.

The value zero shall be set for ERR-RTA-PDU, ACK-RTA-PDU, and NACK-RTA-PDU but the receiver shall not check it.

Bit 5 – 7: AddFlags.reserved
This field shall be set according to 3.6.3.2.

4.7.2.1.5 Coding of the field SendSeqNum

This field shall be coded as data type Unsigned16 with the following values:

Hexadecimal (0x0000 – 0x7FFF)
contains a valid SendSeqNum of a DATA-RTA-PDU

This field contains the number of the DATA-RTA-PDU. The first issued DATA-RTA-PDU shall contain the SendSeqNum 0. The incrementing and comparison of this number is done using modulo 2^{15} operations.

Hexadecimal (0xFFFE, 0xFFFF)
These values are used to synchronize sender and receiver after establishment of the application relationship. The value 0xFFFF indicates the first DATA-RTA-PDU. The value 0xFFFE indicates that there was no reception of a DATA-RTA-PDU before.

NOTE The first DATA-RTA-PDU sets SendSeqNum=0xFFFF and AckSeqNum=0xFFFE. It is acknowledged with SendSeqNum=0xFFFE and AckSeqNum=0xFFFF. The second DATA-RTA-PDU sets SendSeqNum=0 and AckSeqNum=0xFFFE. The synchronization is necessary because the acyclic protocol does not define any connection monitoring.

4.7.2.1.6 Coding of the field AckSeqNum

This field shall be coded as data type Unsigned16 with the following values:

Hexadecimal (0x0000 – 0x7FFF)
contains a valid AckSeqNum

This field contains the number of the DATA-RTA-PDU that is expected to be acknowledged or which is acknowledged.

Hexadecimal (0xFFFF, 0xFFFE)
contains an initial AckSeqNum

The value 0xFFFE indicates that there was no DATA-RTA-PDU received before. The value 0xFFFF indicates acknowledges the reception of the first DATA-RTA-PDU.

4.7.2.1.7 Coding of the field VarPartLen

This field shall be coded as data type Unsigned16 with the following values:

Decimal (0 – 1 432)
contains the number of octets of the following user data

The value 0 shall not be used within a DATA-RTA-PDU.

The value shall be 18 if a DATA-RTA-PDU contains an Alarm-Ack-PDU.

The value shall be 22 if a DATA-RTA-PDU contains an Alarm-Notification-PDU without additional data.

The value shall be in the range from 25 to 1 432 if a DATA-RTA-PDU contains an Alarm-Notification-PDU with data.

The value 4 shall be used within an ERR-RTA-PDU.

The value 0 shall be used within an ACK-RTA-PDU and a NACK-RTA-PDU.

NOTE Padding octets does not affect the field VarPartLen.

4.7.3 Application Relationship Protocol Machines (ARPMs)

4.7.3.1 APMS

4.7.3.1.1 Primitive definitions

4.7.3.1.1.1 Primitives exchanged between APMS and APMS user

The service primitives including their associated parameters issued by APMS user received by APMS and vice versa are described in the RTA ASE in the service definition.

4.7.3.1.1.2 Primitives exchanged between APMS and LMPM

The service primitives including their associated parameters issued by APMS received by LMPM and vice versa are described in the Common DL Mapping ASE in the service definition.

4.7.3.1.1.3 Parameters of APMS primitives

The parameters used with the primitives exchanged between the APMS user and the APMS are described in FAL Service Definition.

4.7.3.1.2 State machine description

The CLOSED state indicates that an initialisation is needed. Activate service sets the machine in the OPEN state waiting for an A-Data service request. After issuing the transmission of data the WACK state is used to wait for an acknowledge. Receiving the acknowledge will set back the machine to the OPEN state. A Close service request resets the protocol machine to the CLOSED state.

Local variables of the APMS

PDU
A local variable representing the UDP-RTA-PDU or the RTA-PDU depending on the transport. Used to show the protocol specific setup and validation of RTA fields. The setup and validation of additional PDU-Parameters (e.g. IP-header) is not shown.

PDUType: 1 = DATA, 2 = NACK, 3 = ACK, 4 = ERR

Seq_Count (Unsigned 16)
This local variable contains the counter value which shall be used at the conveyance of the next DATA-RTA-PDU.

Seq_CountO (Unsigned 16)
This local variable contains the previous counter value of the Seq_Count variable.

Retry (Unsigned8)
This local variable is loaded with MRetry and decremented at every retransmission of a frame. A value of zero indicates the failure in that transaction. A new Activate service is needed to re-establish the communication between the peer providers.

TimeAct (Boolean)
This local variable will be used to find the first full timer cycle after transmitting or receiving a PDU.

4.7.3.1.3 APMS state table

Table 136 contains the complete description of the APMS state machine. A LMPM_A_Data.ind primitive will be accepted if the value of the PDUType.Type field is DATA-RTA-PDU, ACK-RTA-PDU or NACK-RTA-PDU and the PDUType. Version is RTA_VERS=1.

The APMS state machine specify the abort sequence and shows the corresponding ERR-RTA-PDU. These PDU shall only be sent from the APMS for low priority alarms from both sides of the connection.

Table 136 – APMS state table

#	Current State	Event /Condition =>Action	Next State
1	CLOSED	**APMS_Activate.req(CREP,DA,SA, FrameID,VLANPrio, VLANID, RTATimeoutFactor, Mretry,Transport, DstIP, SrcIP)** => Seq_Count:=0xffff Seq_CountO:=0xfffe Store DA,SA,FrameID,VLANPrio, VLAN, Mretry, Transport, DstIP, SrcIP A_Timer_add.req (CREP,RTATimeoutFactor) APMS_Activate.cnf (+)(CREP)	OPEN
2	CLOSED	**APMS_Close.req(CREP)** => APMS_Close.cnf (+)(CREP)	CLOSED
3	CLOSED	**APMS_A_Data.req(CREP, Data)** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_A_Data.cnf(-)(CREP,ERRCLS,ERRCODE)	CLOSED
4	CLOSED	**A_Data_ind** => ignore	CLOSED
5	CLOSED	**A_Data_cnf** => ignore	CLOSED
6	OPEN	**APMS_Activate.req(CREP,DA,SA, FrameID,VLANPrio, VLANID, RTATimeoutFactor, MRetry, Transport, DstIP, SrcIP)** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	OPEN
7	OPEN	**APMS_Close.req(CREP,ErrCode)** /VLANPrio == Low => PDU.Version:=1 PDU.PDUType:=ERR PDU.AddFlags.TACK:=0 PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.PNIOStatus.ErrorCode = 0xCF PDU.PNIOStatus.ErrorDecode = 0x81 PDU.PNIOStatus.ErrorCode1 = 0xFD PDU.PNIOStatus.ErrorCode2 = ErrCode A_Data_req A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED

#	Current State	Event /Condition =>Action	Next State
8	OPEN	**APMS_Close.req(CREP)** /VLANPrio == High => A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
9	OPEN	**APMS_A_Data.req(CREP, Data)** =>PDU.Version:=1 PDU.PDUType:=DATA PDU.AddFlags.TACK:=Tack PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.RTA-SDU:=DataStore PDU.Flags PDU.RTA-SDURetry:= M_Retry+1 A_Data_req	WACK
10	OPEN	**A_Data_ind** /PDU.PDUType == DATA || PDU.PDUType == NAK || PDU.PDUType == ACK => ignore	OPEN
11	OPEN	**A_Data_ind** /PDU.Type == ERR => ERRCLS := PDU.PNIOStatus.ErrorCode1 ERRCODE := PDU.PNIOStatus.ErrorCode2 APMS_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN
12	OPEN	**A_Data_cnf** => ignore	OPEN
13	OPEN	**A_Timer_event.ind(CREP)** => ignore	OPEN
14	WACK	**APMS_Activate.req(CREP,DA,SA, FrameID,VLANPrio, VLANID, RTATimeoutFactor, MRetry, Transport, DstIP, SrcIP)** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_Activate.cnf (-)(CREP,ERRCLS,ERRCODE)	WACK
15	WACK	**APMS_Close.req(CREP,ErrCode)** /VLANPrio == Low => PDU.Version:=1 PDU.PDUType:=ERR PDU.AddFlags:=0 PDU.Window:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.PNIOStatus.ErrorCode = 0xCF PDU.PNIOStatus.ErrorDecode = 0x81 PDU.PNIOStatus.ErrorCode1 = 0xFD PDU.PNIOStatus.ErrorCode2 = ErrCode A_Data_req A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
16	WACK	**APMS_Close.req(CREP)** /VLANPrio == High => A_Timer_remove.req(CREP) APMS_Close.cnf (+)(CREP)	CLOSED
17	WACK	**APMS_A_Data.req(CREP, Data)** => ERRCLS := APMS ERRCODE := INVALID_STATE APMS_A_Data.cnf(-)(CREP,ERRCLS,ERRCODE)	WACK

#	Current State	Event /Condition =>Action	Next State
18	WACK	**A_Data_ind** /(PDU.PDUType == DATA || PDU.PDUType == ACK) && PDU.AckSeqNum == Seq_Count => Seq_CountO=Seq_Count Seq_Count:= (Seq_Count+1) & 0x7fff APMS_A_Data.cnf(+)(CREP)	OPEN
19	WACK	**A_Data_ind** /(PDU.PDUType == DATA || PDU.PDUType == ACK) && PDU.AckSeqNum == Seq_CountO => ignore	WACK
20	WACK	**A_Data_ind** /(PDU.PDUType == DATA || PDU.PDUType == ACK) && PDU.AckSeqNum != Seq_Count && PDU.AckSeqNum != Seq_CountO => ignore	WACK
21	WACK	**A_Data_ind** /PDU.Type == ERR => APMS_Error.ind(CREP,ERRCLS,ERRCODE)	WACK
22	WACK	**A_Data_ind** /PDU.PDUType == NAK => ignore	WACK
23	WACK	**A_Data_cnf** /LMPM_status == OK => ignore	WACK
24	WACK	**A_Data_cnf** /LMPM_status != OK => ERRCLS := APMS ERRCODE := LMPM APMS_Error.ind(CREP,ERRCLS,ERRCODE)	WACK
25	WACK	**A_Timer_event.ind(CREP)** / Retry != 0 => PDU.Version:=1 PDU.PDUType:=DATA PDU.AddFlags.TACK:=from Stored PDU.AddFlags PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_Count PDU.AckSeqNum:=Seq_CountO of APMR PDU.RTA-SDU:=from Stored PDU.RTA-SDU Retry:= Retry-1 A_Data_req	WACK
26	WACK	**A_Timer_event.ind(CREP)** /Retry = 0 => ERRCLS := APMS ERRCODE := TIMEOUT APMS_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN

4.7.3.1.4 Functions

Table 137 contains the functions or macros used by the APMS and APMR, their arguments and their descriptions.

Table 137 – Functions used by the APMS and APMR

Name	Function

A_Data_ind	if (Transport == RTC)
	LMPM_A_Data.ind(CREP, S_Port, TStamp, DA, SA, VLANPrio, VLANID, A_SDU)
	PDU := A_SDU
	
	if (Transport == UDP)
	LMPM_N_Data.ind(CREP, DA, SA, VLANPrio, VLANId, N_SDU)
	PDU := N_SDU

A_Data_cnf	if (Transport == RTC) LMPM_A_Data.cnf (CREP, D_Port, TStamp,LMPM_status) if (Transport == UDP) LMPM_N_Data.cnf (CREP, D_Port, TStamp, LMPM_status))

A_Data_req	if (Transport == RTC)
	A_SDU := PDU
	LMPM_A_Data.req (CREP, D_Port := AUTO, DA, SA, VLANPrio, VLANID, A_SDU)
	
	if (Transport == UDP)
	N_SDU := PDU
	LMPM_N_Data.req (CREP, D_Port := AUTO, DA, SA, VLANPrio, VLANID, N_SDU)

A_Timer_add	Add a cyclic timer
A_Timer_event	Signals a cyclic timer event
A_Timer_remove	Removes a cyclic timer

4.7.3.1.4.1 A_Data_ind, A_Data_cnf and A_Data_req

Local macros used to send and receive UDP-RTA-PDU or RTA-PDU depending on RTA or UDP transport. With A_Data_req the D_Port is to AUTO so LMPM will select the proper output port. TStamp, S_Port is not used.

4.7.3.1.4.2 A_Timer_add

This local service will be used to add a cylic timer in order to indicate cyclic timer events (A_Timer_event.ind) with the spezified TimeFactor. The timer events will occure till the timer will be removed with the "A_Timer_remove" service.

Table 138 – A_Timer_add

Parameter name	Req
Argument	M
CREP	M
TimeFactor	M

Argument
The argument shall convey the service specific parameters of the service request.

CREP
This parameter identifies the timer for further use

Attribute Type: Unsigned32

TimeFactor
This parameter spezifies the timer cycle as a multiple of 100 ms

Attribute Type: Unsigned16

Time Base: 100 ms

4.7.3.1.4.3 A_Timer_event

This local service will be called if a timer cycle elapsed.

Table 139 – A_Timer_event

Parameter name	Ind
Argument	M
CREP	M

Argument
The argument shall convey the service specific parameters of the service request.

CREP
This parameter identifies the timer.

Attribute Type: Unsigned32

4.7.3.1.4.4 A_Timer_remove

This local service will be used to remove a cyclic timer. After a timer is removed, no mor "A_Timer_event" occure.

Table 140 – A_Timer_remove

Parameter name	Req
Argument	M
CREP	M

Argument
The argument shall convey the service specific parameters of the service request.

CREP
This parameter identifies the timer to be removed.

Attribute Type: Unsigned32

4.7.3.2 APMR

4.7.3.2.1 Primitive definitions

4.7.3.2.1.1 Primitives exchanged between APMR and APMR user

The service primitives including their associated parameters issued by APMR user received by APMR and vice versa are described in the RTA ASE in the service definition.

4.7.3.2.1.2 Primitives exchanged between APMR and LMPM

The service primitives including their associated parameters issued by APMR received by LMPM and vice versa are described in the Common DL Mapping ASE in the service definition.

4.7.3.2.1.3 Parameters of APMR primitives

The parameters used with the primitives exchanged between the APMR user and the APMR are described in FAL Service Definition.

4.7.3.2.2 State machine description

The CLOSED state indicates that an initialisation is needed. The Activate service sets the machine in the OPEN state waiting for LMPM_A_Data service indication. After passing a Data indication to the user the WACK state is used to wait for acknowledge from the APMR user. Receiving acknowledge will set back the machine to the OPEN state. A Close service request sets the protocol machine to the CLOSED state.

Local variables of the APMR

PDU
Local variable representing the UDP-RTA-PDU or RTA-PDU depending on transport. Used to show the protocol specific setup and validation of RTA fields. The setup and validation of additional PDU-Parameters (e.g. IP-header) is not shown.

PDUType: 1 = DATA, 2 = NACK, 3 = ACK, 4 = ERR

Seq_Count (Unsigned 16)
This local variable contains the counter value which shall be used at the receipt of the next DATA PDU.

Seq_CountO (Unsigned 16)
This local variable contains the previous counter value of the Seq_Count variable.

Retry (Unsigned16)
This local variable counts the remaining timer ticks till a transmit retry.

4.7.3.2.3 APMR state table

Table 141 contains the complete description of the APMR state machine. A LMPM_A_Data.ind primitive will be accepted:

- If PDUType.Type field is DATA-RTA-PDU and the PDUType.Version field is 1 and the VarPartLen is greater then 0 and the AddFlagsWindow is 1.
- If PDUType.Type field is ERR-RTA-PDU and the PDUType.Version field is 1 and the VarPartLen is 4.

Table 141 – APMR state table

#	Current State	Event /Condition =>Action	Next State
1	CLOSED	**APMR_Activate.req(CREP,DA,SA,FrameID,VLANPrio, VLANID, Transport, DstIP, SrcIP)** => Seq_Count:=0xffff Seq_CountO:=0xfffe Store DA,SA,FrameId,VLANPrio, VLAN,Mretry,Transport, DstIP, SrcIP APMR_Activate.cnf (+)(CREP)	OPEN
2	CLOSED	**APMR_Close.req(CREP)** => APMR_Close.cnf (+)(CREP)	CLOSED
3	CLOSED	**APMR_ACK.req(CREP)** => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_ACK.cnf(-) (CREP,ERRCLS,ERRCODE)	CLOSED
4	CLOSED	**A_Data_ind** => ignore	CLOSED
5	CLOSED	**A_Data_cnf** => ignore	CLOSED
6	OPEN	**APMR_Activate.req(CREP,DA,SA,FrameID,VLANPrio, VLANID, Transport, DstIP, SrcIP)** => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	OPEN
7	OPEN	**APMR_Close.req(CREP)** => APMR_Close.cnf (+)(CREP)	CLOSED

#	Current State	Event /Condition =>Action	Next State
8	OPEN	**APMR_ACK.req(CREP)** => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_ACK.cnf(-) (CREP,ERRCLS,ERRCODE)	OPEN
9	OPEN	**A_Data_ind** /PDU.Type == ERR => ERRCLS := PDU.PNIOStatus.ErrorCode1 ERRCODE := PDU.PNIOStatus.ErrorCode2 APMR_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN
10	OPEN	**A_Data_ind** /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum == Seq_Count => Data := PDU.RTA-SDU APMR_A_Data.ind(CREP, Data)	WACK
11	OPEN	**A_Data_ind** /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum == Seq_CountO =>PDU.Version:=1 PDU.Type:= ACK PDU.AddFlags.TACK:=0 PDU.AddFlags.WindowSize:=1 PDU.SendSeqNum:=Seq_CountO of APMS PDU.AckSeqNum:=Seq_CountO A_Data_req	OPEN
12	OPEN	**A_Data_ind** /PDU.Type == DATA && PDU.AddFlags.Tack && PDU.SendSeqNum != Seq_Count && PDU.SendSeqNum != Seq_CountO => ERRCLS := RTA_ERR_CLS_PROTOCOL ERRCODE := RTA_ERR_CODE_SEQ PDU.Version:=1 PDU.Type:= NAK PDU.AddFlags.TACK:=0 PDU.AddFlags.Window:=1 PDU.SendSeqNum:=Seq_CountO of APMS PDU.AckSeqNum:=Seq_CountO A_Data_req	OPEN
13	OPEN	**A_Data_ind** /(PDU.Type == DATA && !PDU.AddFlags.Tack) => ignore	OPEN
14	OPEN	**A_Data_ind** /PDU.Type == ACK || PDU.Type == NAK => ignore	OPEN
15	OPEN	**A_Data_cnf** /LMPM_status == OK => ignore	OPEN
16	OPEN	**A_Data_cnf** /LMPM_status != OK => ERRCLS := APMR ERRCODE := LMPM APMR_Error.ind(CREP,ERRCLS,ERRCODE)	OPEN
17	WACK	**APMR_Activate.req(CREP,DA,SA,FrameID,VLANPrio, VLANID, Transport, DstIP, SrcIP)** => ERRCLS := APMR ERRCODE :=INVALID_STATE APMR_Activate.cnf (-) (CREP,ERRCLS,ERRCODE)	WACK
18	WACK	**APMR_Close.req(CREP)** => APMR_Close.cnf (+)(CREP)	CLOSED

#	Current State	Event /Condition =>Action	Next State
19	WACK	**APMR_ACK.req(CREP)** =>Seq_CountO :=Seq_Count Seq_Count := (Seq_Count+1) & 0x7fff PDU.Version :=1 PDU.Type := ACK PDU.AddFlags.TACK :=0 PDU.AddFlags.WindowSize :=1 PDU.SendSeqNum :=Seq_CountO of APMS PDU.AckSeqNum :=Seq_CountO A_Data_req APMR_ACK.cnf(+)(CREP)	OPEN
20	WACK	**A_Data_ind** /PDU.Type != ERR => ignore	WACK
21	WACK	**A_Data_ind** /PDU.Type == ERR => APMR_Error.ind(CREP,ERRCLS,ERRCODE)	WACK
22	WACK	**A_Data_cnf** /LMPM_status == OK => ignore	WACK
23	WACK	**A_Data_cnf** /LMPM_status != OK => ERRCLS := APMR ERRCODE := LMPM APMR_Error.ind(CREP,ERRCLS,ERRCODE)	WACK

4.7.3.2.4 Functions

Table 137 contains the functions or macros used by the APMS and APMR, their arguments and their descriptions.

4.8 Remote procedure call

4.8.1 RPC syntax description

4.8.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.8.1.2 RPC APDU abstract syntax

Table 142 shows the utilization of the Publication C706 of The Open Group and defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 142 shall represent the content of the DLSDU in Table 4.

Table 142 – RPC APDU syntax

APDU name	APDU structure
CL-RPC-PDU	IPHeader, UDPHeader, RPCHeader, NDRDataRequest ^ NDRDataResponse ^ NDREPMapLookupReq ^ NDREPMapLookupRes ^ NDREPMapLookupFreeReq ^ NDREPMapLookupFreeRes ^ NDRAck ^ NDRQuAck^NDRQuit ^ NDRFack ^ NDRFault ^ NDRNoCall ^ NDRWorking ^ NDRPing ^ NDRReject

Table 143 defines structures for substitutions of elements of the APDU structures shown in Table 142.

Table 143 – RPC substitutions

Substitution name	Structure
RPCHeader	RPCVersion (4), RPCPacketType, RPCFlags, RPCFlags2, RPCDRep, RPCSerialHigh, RPCObjectUUID [a], RPCInterfaceUUID [b], RPCActivityUUID, RPCServerBootTime, RPCInterfaceVersion, RPCSequenceNmb, RPCOperationNmb [c], RPCInterfaceHint, RPCActivityHint, RPCLengthOfBody, RPCFragmentNmb, RPCAuthenticationProtocol, RPCSerialLow
NDRDataRequest	ArgsMaximum, ArgsLength, MaximumCount, Offset(0), ActualCount, PROFINETIOServiceReqPDU
NDRDataResponse	PNIOStatus, ArgsLength, MaximumCount, Offset(0), ActualCount, [PROFINETIOServiceResPDU]
NDRFault	RPCNCAFaultStatus
NDRAck	NULL
NDRQuAck	NULL ^ (RPCCancelVersion(0), RPCCancelID, RPCServerIsAccepting)
NDRQuit	NULL ^ (RPCCancelVersion(0), RPCCancelID)
NDRFack	NULL ^ (RPCVersionFack(0), RPCPad1, RPCWindowSize, RPCMaxTsdu, RPCMaxFragSize, RPCSerialNumber, RPCSelAckLen, RPCArrayOfSelAck*)
NDRNoCall	NULL ^ NDRFack
NDRWorking	NULL
NDRPing	NULL
NDRReject	RPCNCARejectStatus
NDREPMapLookupReq	RPCInquiryType, RPCObjectReference(1), RPCObjectUUID, RPCInterfaceReference(2), RPCInterfaceUUID, RPCInterfaceVersionMajor, RPCInterfaceVersionMinor, RPCVersionOption(1), RPCEntryHandleAttribute(0), RPCEntryHandleUUID, RPCMaxEntries
NDREPMapLookupRes	RPCEntryHandleAttribute(0), RPCEntryHandleUUID, RPCNumberOfEntries, RPCMaxEntries, RPCEntriesOffset, RPCEntriesCount, [(RPCObjectUUID, RPCTowerReference, RPCAnnotationOffset, RPCAnnotationLength, RPCAnnotation, [RPCGap*], RPCTowerLength, RPCTowerOctetStringLength, [RPCTowerOctetString*], [RPCGap*])*], RPCEPMapStatus
RPCTowerOctetString	RPCFloorCount, [RPCFloor*]
RPCFloor	RPCLHSByteCount, [RPCProtocolIdentifierString*], RPCRHSByteCount, [RPCRelatedData*]
RPCProtocolIdentifierString	(RPCID, RPCInterfaceUUID, RPCInterfaceVersionMajor) ^ (RPCID, RPCDataRepresentationUUID, RPCInterfaceVersionMajor) ^ RPCProtocolIdentifier ^ RPCServerUDPPort ^ RPCHostAddress
RPCRelatedData	RPCInterfaceVersionMinor ^ RPCPortNumber ^ RPCIPAddress
NDREPMapLookupFreeReq	RPCEntryHandleAttribute, RPCEntryHandleUUID
NDREPMapLookupFreeRes	RPCEntryHandleAttribute, RPCEntryHandleUUID, RPCEPMapStatus
RPCAnnotation	DeviceType, Blank, OrderID, Blank, HWRevision, Blank, SWRevisionPrefix, SWRevision, EndTerm

[a] To identify IO device, IO controller, IO supervisor.

[b] To identify PNIO interface type.

[c] To identify the service type e.g. Connect, Release, Read, Write, Control.

4.8.2 RPC Transfer syntax

4.8.2.1 Coding section related to CL-RPC-PDU

4.8.2.1.1 Coding of the field RPCVersion

This field shall be coded as data type Unsigned8. The value shall be 4.

4.8.2.1.2 Coding of the field RPCPacketType

This field shall be coded as data type Unsigned8.

The values shall be encoded according to Table 144.

Table 144 – RPCPacketType

Value (hexadecimal)	Meaning
0x00	Request
0x01	Ping
0x02	Response
0x03	Fault
0x04	Working
0x05	No call, response to ping
0x06	Reject
0x07	Acknowledge
0x08	Connectionless cancel
0x09	Fragment acknowledge (FACK-PDU)
0x0A	Cancel acknowledge
0x0B – 0xFF	Reserved

The decoder shall only check the 5 least significant bits.

4.8.2.1.3 Coding of the field RPCFlags

This field shall be coded as data type according to 3.7.3.3.

The values shall be encoded according to Table 145.

Table 145 – RPCFlags

Bit	Meaning if set to 1
Bit 0	Implementation specific, shall be set 0
Bit 1	Last fragment
Bit 2	Fragment
Bit 3	No fragment acknowledge requested
Bit 4	Maybe
Bit 5	Idempotent
Bit 6	Broadcast
Bit 7	Implementation specific, shall be set to 0

4.8.2.1.4 Coding of the field RPCFlags2

This field shall be coded as data type according to 3.7.3.3.

The values shall be encoded according to Table 146.

Table 146 – RPCFlags2

Bit	Meaning if set to 1
Bit 0	Implementation specific, shall be set to 0
Bit 1	Cancel was pending at call end
Bit 2	Reserved
Bit 3	Reserved
Bit 4	Reserved
Bit 5	Reserved
Bit 6	Reserved
Bit 7	Reserved

4.8.2.1.5 Coding of the field RPCDRep

This field shall be coded as data type OctetString[3].

RPCDRep Octet 1
The coding of the first octet shall be according to 3.7.3.3 and the individual bits shall have the meaning defined in Table 147.

Table 147 – RPCDRep.Character- and IntegerEncoding

Bit	Field Name	Value	Meaning
0 – 3	RPCDRep.CharacterEncoding [a]	0	ASCII
		1	EBCDIC
4 – 7	RPCDRep.IntegerEncoding [b]	0	Big endian
		1	Little endian

[a] Not used within Type 10.

[b] As an exception to 3.7.3.5 the RPC encoding rules are applied to the RPCHeader, NDR substitutions and NDREPMap substitutions within Type 10. It only uses Unsigned8, Unsigned16, or Unsigned32 there. The user data of the Type 10 service definitions are not influenced because everything is an opaque OctetString there.

RPCDRep Octet 2
The values of the second octet shall be encoded according to Table 148.

Table 148 – RPCDRep Octet 2 – Floating Point Representation

Value (hexadecimal)	Meaning
0x00	IEEE
0x01	VAX
0x02	CRAY
0x03	IBM
0x04 – 0xFF	Reserved

RPCDRep Octet 3
The value of the third octet shall be zero.

4.8.2.1.6 Coding of the field RPCSerialHigh

This field shall be coded as data type Unsigned8. The value contains the high byte of the fragment number of the call.

4.8.2.1.7 Coding of the field RPCSerialLow

This field shall be coded as data type Unsigned8. The value contains the low byte of the fragment number of the call.

NOTE The value of these fields is incremented which each transmission even with a transmission repetition in distinction to the field FragmentNmb.

4.8.2.1.8 Coding of the field RPCObjectUUID

This field shall be coded as data type structure containing the following elements:

- Data1 as Unsigned32
- Data2 as Unsigned16
- Data3 as Unsigned16
- Data4 as array of eight Unsigned8 (Octet 1 to Octet 8)

The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The ordering of octets of Data4 is shown in Table 149.

Table 149 – RPCObjectUUID.Data4

Octet	Meaning
1	Value see Table 150
2	
3	Instance High
4	Instance Low
5	DeviceID High
6	DeviceID Low
7	VendorID High
8	VendorID Low

Defined values for PNIO items are shown in Table 150.

Table 150 – RPCObjectUUID – defined values

Value	Description	
UUID_IO_ObjectInstance_XYZ DEA00000-6C97-11D1-8271-{xxxxyyyyzzzz}	Identifies a special object instance within a physical device in case there are more than one where	
	xxxx	Represents the instance or node number
	yyyy	Identify the Device ID as a vendor specific number for the device class
	zzzz	Represents the Vendor ID as a central administrative number assigned by the responsible user organisation

4.8.2.1.9 Coding of the field RPCInterfaceUUID

This field shall be coded as data type structure containing the following elements:

- Data1 as Unsigned32
- Data2 as Unsigned16
- Data3 as Unsigned16
- Data4 as array of eight Unsigned8

The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

Defined values for PNIO items are shown in Table 151.

Table 151 – RPCInterfaceUUID – defined values

Value	Description
UUID_IO_DeviceInterface DEA00001-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO device uniquely.
UUID_IO_ControllerInterface DEA00002-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO controller uniquely.
UUID_IO_SupervisorInterface DEA00003-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO supervisor uniquely.
UUID_IO_ParameterServerInterface DEA00004-6C97-11D1-8271-00A02442DF7D	Identifies the interface of an IO parameter server uniquely.
UUID_EPMap_Interface E1AF8308-5D1F-11C9-91A4-08002B14A0FA	Identifies the interface of the endpoint mapper. The RPCInterfaceVersion shall be 0x00030000.

4.8.2.1.10 Coding of the field RPCActivityUUID

This field shall be coded as data type structure containing the following elements:

- Data1 as Unsigned32
- Data2 as Unsigned16
- Data3 as Unsigned16
- Data4 as array of eight Unsigned8

NOTE The response mirrors the content of the field of the request.

The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.11 Coding of the field RPCServerBootTime

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

NOTE Idempotent functions do not need to maintain this value. Only the endpointmapper is allowed to answer with zero.

4.8.2.1.12 Coding of the field RPCInterfaceVersion

This field shall be coded as data type Unsigned32. The value shall be set to 0x00010000. The representation on the wire is according to the value in the field “RPCDRep.IntegerEncoding”.

The major version shall be set to 1 for this version (high word).

The minor version shall be set to 0 for this version (low word).

NOTE Other values may be used for compatible extensions.

4.8.2.1.13 Coding of the field RPCSequenceNmb

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.14 Coding of the field RPCOperationNmb

This field shall be coded as data type Unsigned16. The RPCOperationNmb identifies the PNIO service supported by the PNIO interfaces:

- IO device,
- IO controller, and
- IO supervisor.

The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The values for PNIO services are defined in Table 152.

Table 152 – RPCOperationNmb (IO device, controller and supervisor)

Value (decimal)	Service	Usage
0	Connect	
1	Release	
2	Read	Only valid with ARUUID<>0
3	Write	Only valid with ARUUID<>0
4	Control	
5	Read Implicit	Only valid with ARUUID=0
6 – 65 535	Reserved	

The values for endpoint mapper services according to Table 153 shall be used.

Table 153 – RPCOperationNmb for endpoint mapper

Value (decimal)	Usage	Service
0	Optional	Insert
1	Optional	Delete
2	Mandatory	Lookup
3	Optional	Map
4	Mandatory	LookupHandleFree
5	Optional	InqObject
6	Optional	MgmtDelete
7 – 65 535		Reserved

4.8.2.1.15 Coding of the field RPCInterfaceHint

This field shall be coded as data type Unsigned16. The value shall be set to no hint (0xFFFF) for this version. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.16 Coding of the field RPCActivityHint

This field shall be coded as data type Unsigned16. The value shall be set to no hint (0xFFFF) for this version. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.17 Coding of the field RPCLengthOfBody

This field shall be coded as data type Unsigned16. The value shall be set to the number of octets of NDRData of the current frame. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

NOTE The conveyance of NDRData may require more than one frame. In this case RPCLengthOfBody contains only the number of octets for the current frame.

4.8.2.1.18 Coding of the field RPCFragmentNmb

This field shall be coded as data type Unsigned16. The value shall be set to the number of the current fragment. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.19 Coding of the field RPCAuthenticationProtocol

This field shall be coded as data type Unsigned8. The value shall be set to zero for no authentication.

4.8.2.1.20 Coding of the field RPCCancelVersion

This field shall be coded as data type Unsigned32. The value shall be set to zero. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.21 Coding of the field RPCCancelID

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.22 Coding of the field RPCServerIsAccepting

This field shall be coded as data type Unsigned8.

4.8.2.1.23 Coding of the field RPCVersionFack

This field shall be coded as data type Unsigned8. The value shall be set to zero.

4.8.2.1.24 Coding of the field RPCPad1

This field shall be coded as data type Unsigned8.

4.8.2.1.25 Coding of the field RPCWindowSize

This field shall be coded as data type Unsigned16. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.26 Coding of the field RPCMaxTsdu

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.27 Coding of the field RPCMaxFragSize

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.28 Coding of the field RPCSerialNumber

This field shall be coded as data type Unsigned16. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.29 Coding of the field RPCSelAckLen

This field shall be coded as data type Unsigned16. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.30 Coding of the field RPCArrayOfSelAck

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.1.31 Coding of the field RPCDataRepresentationUUID

This field shall be coded as data type structure containing the following elements:

- Data1 as Unsigned32
- Data2 as Unsigned16
- Data3 as Unsigned16
- Data4 as array of eight Unsigned8

The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

Defined values are shown in Table 154.

Table 154 – RPCDataRepresentationUUID – defined values

Value [UUID]	Value	Description
8A885D04-1CEB-11C9-9FE8-08002B104860	Data representation	Identifies the RPC data representation uniquely.

4.8.2.2 Coding section related to NDREPMapPDU

4.8.2.2.1 Coding of the field DeviceType

This field shall be coded as data type VisibleString[25]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 25 octets.

4.8.2.2.2 Coding of the field OrderID

This field shall be coded as data type VisibleString[20]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 20 octets.

4.8.2.2.3 Coding of the field HWRevision

This field shall be coded as data type VisibleString[5]. The value shall be set manufacturer specific using the character "0"-"9" and "<Blank>". The range is from "<Blank><Blank><Blank><Blank>0" to "99999".

NOTE As an example HWRevision could be "<Blank><Blank><Blank><Blank>2", "<Blank>5714", or "61214".

4.8.2.2.4 Coding of the field SWRevisionPrefix

This field shall be coded as data type VisibleString[1]. The value shall be set manufacturer specific using the character:

- "V" for an officially released version
- "R" for Revision
- "P" for Prototype
- "U" for Under Test (Field Test)
- "T" for Test Device

4.8.2.2.5 Coding of the field SWRevision

This field shall be coded as data type VisibleString[9]. The value shall be set manufacturer specific using the character "0"-"9" and "<Blank>". The range is from "<Blank><Blank>0<Blank><Blank>0<Blank><Blank>0" to "999999999".

NOTE As an example SWRevision could be "<Blank><Blank>1<Blank><Blank>0<Blank><Blank>0". After each group of three octets a "." should be inserted for visualization.

4.8.2.2.6 Coding of the field Blank

This field shall be coded as data type OctetString with 1 octet. The value shall be set to 0x20.

4.8.2.2.7 Coding of the field RPCInquiryType

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The coding shall be according to Table 155.

Table 155 – RPCInquiryType

Value (hexadecimal)	Meaning
0x00000000	Read all registered interfaces with objects. (mandatory)
0x00000001	Read all objects for one registered interface. (optional)
0x00000002	Read all interfaces including a dedicated object. (optional)
0x00000003	Read one dedicated interface with one dedicated object. (optional)
0x00000004 – 0xFFFFFFFF	Reserved

4.8.2.2.8 Coding of the field RPCObjectReference

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 1.

4.8.2.2.9 Coding of the field RPCInterfaceReference

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 2.

4.8.2.2.10 Coding of the field RPCInterfaceVersionMajor

This field shall be coded as data type Unsigned16. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.11 Coding of the field RPCInterfaceVersionMinor

This field shall be coded as data type Unsigned16. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.12 Coding of the field RPCVersionOption

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 1.

4.8.2.2.13 Coding of the field RPCEntryHandleAttribute

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 0.

4.8.2.2.14 Coding of the field RPCEntryHandleUUID

This field shall be coded as data type UUID. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.15 Coding of the field RPCMaxEntries

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set at least to 1.

4.8.2.2.16 Coding of the field RPCNumberOfEntries

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.17 Coding of the field RPCEntriesOffset

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 0.

4.8.2.2.18 Coding of the field RPCEntriesCount

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.19 Coding of the field RPCTowerReference

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 3.

4.8.2.2.20 Coding of the field RPCAnnotationOffset

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to 0.

4.8.2.2.21 Coding of the field RPCAnnotationLength

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set from 0 to 64.

4.8.2.2.22 Coding of the field EndTerm

This field shall be coded as data type Unsigned8. The value shall be set to 0.

4.8.2.2.23 Coding of the field RPCGap

This field shall be coded as data type Unsigned8.

4.8.2.2.24 Coding of the field RPCTowerLength

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.25 Coding of the field RPCTowerOctetStringLength

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

4.8.2.2.26 Coding of the field RPCEPMapStatus

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The coding shall be according to Table 156.

Table 156 – RPCEPMapStatus

Value (hexadecimal)	Meaning
0x00000000	RPC okay
0x16C9A0D6	Endpoint not registered
others	Reserved

4.8.2.2.27 Coding of the field RPCFloorCount

This field shall be coded as data type Unsigned16. The value shall be set to 5. The encoding shall be independently of the field RPCDRep always little endian.

4.8.2.2.28 Coding of the field RPCLHSByteCount

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always little endian.

4.8.2.2.29 Coding of the field RPCRHSByteCount

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always little endian.

4.8.2.2.30 Coding of the field RPCID

This field shall be coded as data type Unsigned8. The value shall be set to 0x0D.

4.8.2.2.31 Coding of the field RPCProtocolIdentifier

This field shall be coded as data type Unsigned8. The value shall be set to 0x0A.

4.8.2.2.32 Coding of the field RPCServerUDPPort

This field shall be coded as data type Unsigned8. The value shall be set to 0x08.

4.8.2.2.33 Coding of the field RPCHostAddress

This field shall be coded as data type Unsigned8. The value shall be set to 0x09.

4.8.2.2.34 Coding of the field RPCPortNumber

This field shall be coded as data type Unsigned16. The encoding shall be independently of the field RPCDRep always big endian.

4.8.2.2.35 Coding of the field RPCIPAddress

This field shall be coded as data type Unsigned32. The encoding shall be independently of the field RPCDRep always big endian.

4.8.2.3 Coding section related to NDRData

4.8.2.3.1 Coding of the field ArgsMaximum

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to the maximum buffer size available for the response.

4.8.2.3.2 Coding of the field ArgsLength

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to the number of octets within the PROFINETIOServiceReqPDU or PROFINETIOServiceResPDU.

4.8.2.3.3 Coding of the field MaximumCount

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

In case of a request the value shall be set to the same value as in the field ArgsMaximum. Within the response the value shall be taken from the field ArgsMaximum of the appropriate request.

4.8.2.3.4 Coding of the field Offset

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to zero.

4.8.2.3.5 Coding of the field ActualCount

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian).

The value shall be set to the same value as in the field ArgsLength.

4.8.2.3.6 Coding of the field PNIOStatus

This field shall be coded as data type Unsigned32. The byte ordering shall be according to the value of the field RPCDRep (little endian or big endian) within the first field of the NDRDataResponse. In all other cases the byte ordering shall be big endian.

The content is defined in 6.2.4.68. The PNIOStatus shall be calculated according the following equation.

$$\begin{aligned} \text{PNIOStatus} = \; & \text{ErrorCode} \times 16\,777\,216 \; + \\ & \text{ErrorDecode} \times 65\,536 \; + \\ & \text{ErrorCode1} \times 256 \; + \\ & \text{ErrorCode2} \end{aligned} \qquad (47)$$

4.8.2.4 Coding section related to RPC (NCA Codes)

4.8.2.4.1 Coding of the field RPCNCAFaultStatus

The RPC specific NCA error codes shall be coded as Unsigned32 with the values according to Table 157.

Table 157 – Values of NCAFaultStatus

Value (hexadecimal)	Definition
0x1C000001	NCA_s_fault_int_div_by_zero
0x1C000002	NCA_s_fault_addr_error
0x1C000003	NCA_s_fault_fp_div_zero
0x1C000004	NCA_s_fault_fp_underflow
0x1C000005	NCA_s_fault_fp_overflow
0x1C000006	NCA_s_fault_invalid_tag
0x1C000007	NCA_s_fault_invalid_bound
0x1C000008	NCA_s_rpc_version_mismatch
0x1C000009	NCA_s_unspec_reject
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_s_who_are_you_failed
0x1C00000C	NCA_s_manager_not_entered
0x1C00000D	NCA_s_fault_chancel
0x1C00000E	NCA_s_fault_ill_inst
0x1C00000F	NCA_s_fault_fp_error
0x1C000010	NCA_s_fault_int_overflow
0x1C000012	NCA_s_fault_unspec
0x1C000013	NCA_s_fault_remote_comm_failure
0x1C000014	NCA_s_fault_pipe_empty
0x1C000015	NCA_s_fault_pipe_closed
0x1C000016	NCA_s_fault_pipe_order
0x1C000017	NCA_s_fault_pipe_discipline
0x1C000018	NCA_s_fault_pipe_comm_error
0x1C000019	NCA_s_fault_pipe_memory
0x1C00001A	NCA_s_fault_context_mismatch
0x1C00001B	NCA_s_fault_remote_no_memory
0x1C00001C	NCA_s_invalid_pres_context_id
0x1C00001D	NCA_s_unsupported_authn_level
0x1C00001F	NCA_s_invalid_checksum
0x1C000020	NCA_s_invalid_crc
0x1C000021	NCA_s_fault_user_defined
0x1C000022	NCA_s_fault_tx_open_failed
0x1C000023	NCA_s_fault_codeset_conv_error
0x1C010001	NCA_s_comm_failure
0x1C010015	NCA_s_fault_string_too_long

4.8.2.4.2 Coding of the field RPCNCARejectStatus

The RPC specific NCA error codes shall be coded as Unsigned32 with the values according to Table 158.

Table 158 – Values of NCARejectStatus

Value (hexadecimal)	Definition
0x1C000008	NCA_rpc_version_mismatch
0x1C000009	NCA_unspec_reject
0x1C00000A	NCA_s_bad_actid
0x1C00000B	NCA_who_are_you_failed
0x1C00000C	NCA_manager_not_entered
0x1C010002	NCA_op_rng_error
0x1C010003	NCA_unk_if
0x1C010006	NCA_wrong_boot_time
0x1C010009	NCA_s_you_crashed
0x1C01000B	NCA_proto_error
0x1C010013	NCA_out_args_too_big
0x1C010014	NCA_server_too_busy
0x1C010017	NCA_unsupported_type
0x1C00001D	NCA_unsupported_authn_level
0x1C00001F	NCA_invalid_checksum
0x1C000020	NCA_invalid_crc

4.8.3 Application Relationship Protocol Machines (ARPMs)

4.8.3.1 RPCPM

4.8.3.1.1 Primitive definitions

4.8.3.1.1.1 Primitives exchanged between RPCPM and RPCPM user

The service primitives including their associated parameters issued by RPCPM user received by RPCPM and vice versa are described in the RPC ASE in the service definition.

4.8.3.1.1.2 Primitives exchanged between RPCPM and LMPM

The service primitives including their associated parameters issued by RPCPM received by LMPM and vice versa are described in the RPC ASE in the service definition.

4.8.3.1.1.3 Parameters of RPCPM primitives

The parameters used with the primitives exchanged between the RPCPM user and the RPCPM are described in FAL Service Definition.

4.8.3.1.2 State machine description

4.8.3.1.3 RPC state table

4.8.3.1.4 Functions

4.8.3.2 Monitoring of services

The Ping service initiated by a service requester is called to monitor the health of the responder. The Cancel service may be repeated up to three times.

Timeout Ping
The value shall be 2 s.

NOTE The ping service is used if the request is completely sended and the response is still pending.

Timeout FRAG
The value shall be 2 s.

NOTE The transmission of a fragment will be repeated up to three times after two seconds if there is no acknowledge.

Timeout Cancel
The value shall be 1 s.

NOTE The Cancel service will be repeated up to three times after one second if there is no reaction.

Timeout Resend, Timeout Ack, Timeout Broadcast, Timeout IDLE, Timeout WAIT
The values don't care.

NOTE The above described parameters are defined with their values in DCE RPC.

4.9 Link layer discovery

4.9.1 FAL common syntax description

4.9.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.9.1.2 LLDP APDU abstract syntax

Table 159 defines the abstract syntax of the LLDP PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 159 shall represent the content of the DLSDU in Table 4.

Table 159 – LLDP APDU syntax

APDU name	APDU structure
LLDP-PDU	LLDPChassis, LLDPPort, LLDPTTL, LLDP-PNIO-PDU, LLDPEnd
LLDP-PNIO-PDU	{ LLDP_PNIO_DELAY [a], LLDP_PNIO_PORTSTATUS, [LLDP_PNIO_ALIAS], LLDP_PNIO_MRPPORTSTATUS [b], LLDP_PNIO_CHASSIS_MAC, LLDP8023MACPHY, LLDPManagement, LLDP_PNIO_PTCPSTATUS [c], [LLDPOption*], [LLDP8021*], [LLDP8023*] }

[a] Shall only exist, if LineDelay measurement is supported.
[b] Shall only exist, if MRP is activated for this port.
[c] Shall only exist, if PTCP is activated.

Table 160 defines structures for substitutions of elements of the APDU structures shown in Table 159.

Table 160 – LLDP substitutions

Substitution name	Structure
LLDPChassis	LLDPChassisStationName ^ LLDPChassisMacAddress [a]
LLDPChassisStationName	LLDP_TLVHeader [b], LLDP_ChassisIDSubType(7) [b], LLDP_ChassisID
LLDPChassisMacAddress	LLDP_TLVHeader [b], LLDP_ChassisIDSubType(4) [b], (CMResponderMacAdd ^ CMInitiatorMacAdd) [d]
LLDPPort	LLDP_TLVHeader [b], LLDP_PortIDSubType(7) [b], LLDP_PortID [c]
LLDPTTL	LLDP_TLVHeader [b], LLDP_TimeToLive(20) [b]
LLDP_PNIO_Header	LLDP_TLVHeader [b], LLDP_OUI(00-0E-CF)
LLDP_PNIO_DELAY	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x01), PTCP_PortRxDelayLocal, PTCP_PortRxDelayRemote PTCP_PortTxDelayLocal, PTCP_PortTxDelayRemote, CableDelayLocal
LLDP_PNIO_PORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x02), RTClass2_PortStatus, RTClass3_PortStatus
LLDP_PNIO_ALIAS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x03), LLDP_PortID, LLDP_POINT, LLDP_ChassisID
LLDP_PNIO_MRPPORTSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x04), MRP_DomainUUID, MRRT_PortStatus
LLDP_PNIO_CHASSIS_MAC	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x05), (CMResponderMacAdd ^ CMInitiatorMacAdd) [d]
LLDP_PNIO_PTCPSTATUS	LLDP_PNIO_Header, LLDP_PNIO_SubType(0x06), PTCP_MasterSourceAddress [e], PTCP_SubdomainUUID [f], IRDataUUID [g], LLDP_LengthOfPeriod [g], LLDP_RedPeriodBegin [g], LLDP_OrangePeriodBegin [g], LLDP_GreenPeriodBegin [g]
LLDPEnd	LLDP_TLVHeader [b](0)
LLDP8021	LLDP_TLVHeader [b], LLDP_OUI(00-80-C2) [h], LLDP_8021_SubType [h], Data [h]
LLDP8023	LLDP_TLVHeader [b], LLDP_OUI(00-12-0F) [i], LLDP_8023_SubType [i], Data [i]
LLDP8023MACPHY	LLDP_TLVHeader [b], LLDP_OUI(00-12-0F) [i], LLDP_8023_SubType(1) [i], LLDP_8023_AUTONEG [i], LLDP_8023_PMDCAP [i], LLDP_8023_OPMAU [i]
LLDPManagement	LLDP_TLVHeader [b], LLDP_ManagementData [j]

[a] LLDPChassisMacAddress shall be used if no NameOfStation is assigned.

[b] The encoding of the fields shall be according to IEEE 802.1AB-2005.

[c] Shall be 8 or 14.

[d] Shall be the interface MAC address.

[e] Shall be set zero, if unknown.

[f] Shall be PTCP_SubdomainUUID of PTCP_SyncID := 0. Otherwise the value shall be zero.

[g] Shall be set zero, if unknown.

[h] Shall be set according to IEEE 802.1AB-2005 Annex F.

[i] Shall be set according to IEEE 802.1AB-2005 Annex G.

[j] Shall be set according to IEEE 802.1AB-2005, 9.5.9. It is recommended to set the object identifier according to 4.15.2.

NOTE There are different kinds of MAC addresses. The port MAC address used as SourceAddress and the interface MAC address used as CMResponderMacAdd or CMInitiatorMacAdd.

4.9.2 LLDP transfer syntax

4.9.2.1 General

As an extension to IEEE 802.1AB-2005, 10.5.3.1 this standard defines that value of the field txDelayWhile shall be zero. In this case, every node transmits LLDP PDUs on data change.

4.9.2.2 Coding of the field LLDP_ChassisID

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.32.1.

NOTE The field LLDP_ChassisID is not terminated by zero.

4.9.2.3 Coding of the field LLDP_PortID

This field shall be coded as data type OctetString with 8 or 14 octets.

This field contains the name of the local port which shall be used as Port ID subtype locally assigned compliant with IEEE 802.1AB. The value shall be “port-xyz” or “port-xyz-rstuv” where x, y, z is in the range “0”-“9” from 001 up to 255 and r, s, t, u, v is in the range “0”-“9” from 00000 up to 65535. The values x, y, z shall be used to identify the port number. The values r, s, t, u, v shall be used to identify the slot which contains the port.

The values “port-001-00000” shall be used for the first port submodule for an interface within slot 0 if any other slot may contain another port submodul.

The values “port-001” shall be used for the first port submodule for an interface if no other slot may contain another port submodul.

Furthermore, the definition of RFC 3490 shall be applied.

NOTE The field LLDP_PortID is not terminated by zero.

4.9.2.4 Coding of the field LLDP_PNIO_SubType

This field shall be coded as data type Unsigned8 and shall be set according to Table 161.

Table 161 – LLDP_PNIO_SubType

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Measured delay values
0x02	Port Status
0x03	Alias
0x04	MRP Port Status
0x05	Interface MAC address
0x06	PTCP Status
0x07 – 0xFF	Reserved

4.9.2.5 Coding of the field PTCP_PortRxDelayLocal

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 162.

Table 162 – PTCP_PortRxDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Local RX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.9.2.6 Coding of the field PTCP_PortRxDelayRemote

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 163.

Table 163 – PTCP_PortRxDelayRemote

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Remote RX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.9.2.7 Coding of the field PTCP_PortTxDelayLocal

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 164.

Table 164 – PTCP_PortTxDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Local TX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.9.2.8 Coding of the field PTCP_PortTxDelayRemote

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 165.

Table 165 – PTCP_PortTxDelayRemote

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x00000FFF	Remote TX port delay
0x00001000 – 0xFFFFFFFF	Reserved

4.9.2.9 Coding of the field CableDelayLocal

This field shall be coded as data type Unsigned32, the time base shall be 1 ns and shall be set according to Table 166.

Table 166 – CableDelayLocal

Value (hexadecimal)	Meaning
0x00	Unknown
0x01 – 0x000FFFFF	Measured cable delay
0x00100000 – 0xFFFFFFFF	Reserved

4.9.2.10 Coding of the field RTClass2_PortStatus

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: RTClass2_PortStatus.State
This field shall be set according to the Table 167.

Table 167 – RTClass2_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used
0x01	SYNCDATA_LOADED	Configured
0x02	RTCLASS2_UP	ORANGE Phase activated for transmission and reception of RTClass2 Frames
0x03	Reserved	

Bit 2 – 15: RTClass2_PortStatus.reserved
This field shall be set according to 3.7.3.2.

4.9.2.11 Coding of the field RTClass3_PortStatus

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 2: RTClass3_PortStatus.State
This field shall be set according to the Table 168.

Table 168 – RTClass3_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used
0x01	IRDATA_LOADED	Configured
0x02	RTCLASS3_UP	RED Phase activated for transmission of RTClass3 Frames. Expected next state: RTCLASS3_RUN
0x03	RTCLASS3_DOWN	RED Phase activated for transmission of RTClass3 Frames. Expected next state: IRDATA_LOADED or OFF
0x04	RTCLASS3_RUN	RED Phase activated for transmission and reception of RTClass3 Frames
0x05 – 0x07	Reserved	

Bit 3 – 14: RTClass3_PortStatus.reserved
This field shall be set according to 3.7.3.2.

Bit 15: RTClass3_PortStatus.Mode
This field shall be set according to the Table 169.

Table 169 – RTClass3_PortStatus.Mode

Value (hexadecimal)	Meaning	Usage
0x00	STANDARD	Use all states of RT_CLASS_3 port state machine
0x01	OPTIMIZED	Use optimized states of RT_CLASS_3 port state machine

4.9.2.12 Coding of the field MRRT_PortStatus

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: MRRT_PortStatus.State
This field shall be set according to the Table 170.

Table 170 – MRRT_PortStatus.State

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Not used / not configured
0x01	MRRT_CONFIGURED	Configured
0x02	MRRT_UP	Activated and running
0x03	Reserved	

Bit 2 – 15: MRRT_PortStatus.reserved
This field shall be set according to 3.7.3.2.

4.9.2.13 Coding of the field LLDP_RedPeriodBegin

As defined for RT_CLASS_3 the usage of a port is divided into different time periods. Also the usage of a port is divided into transmit and receive direction. The coding of this field shall be according to 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_RedPeriodBegin.Offset
This field shall be set according to the Table 171.

Table 171 – LLDP_RedPeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Offset relative to the begin of the cycle in nanoseconds	Begin of the RT_CLASS_3 period of the receive direction of the port.
0x003D0900 – 0x7FFFFFFF	Reserved	

Bit 31: LLDP_RedPeriodBegin.Valid
This optional field shall be set according to the Table 172.

Table 172 – LLDP_RedPeriodBegin.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	The value of LLDP_RedPeriodBegin.Offset is not valid. It shall be set to zero.
0x01	Valid	The value of LLDP_RedPeriodBegin.Offset is valid.

4.9.2.14 Coding of the field LLDP_OrangePeriodBegin

As defined for RT_CLASS_2 the usage of a port is divided into different time periods. Also the usage of a port is divided into transmit and receive direction. The coding of this field shall be according to 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_OrangePeriodBegin.Offset
This field shall be set according to the Table 173.

Table 173 – LLDP_OrangePeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Offset relative to the begin of the cycle in nanoseconds	Begin of the RT_CLASS_2 period of the receive direction of the port.
0x003D0900 – 0x7FFFFFFF	Reserved	

Bit 31: LLDP_OrangePeriodBegin.Valid
This optional field shall be set according to the Table 174.

Table 174 – LLDP_OrangePeriodBegin.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	The value of LLDP_OrangePeriodBegin.Offset is not valid. It shall be set to zero.
0x01	Valid	The value of LLDP_OrangePeriodBegin.Offset is valid.

4.9.2.15 Coding of the field LLDP_GreenPeriodBegin

As defined for RT_CLASS_1, RT_CLASS_UDP and the other protocols the usage of a port is divided into different time periods. Also the usage of a port is divided into transmit and receive. The coding of this field shall be according to 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_GreenPeriodBegin.Offset
This field shall be set according to the Table 175.

Table 175 – LLDP_GreenPeriodBegin.Offset

Value (hexadecimal)	Meaning	Usage
0x00000000 – 0x003D08FF	Offset relative to the begin of the cycle in nanoseconds	Begin of the unrestricted period of the receive direction of the port.
0x003D0900 – 0x7FFFFFFF	Reserved	

Bit 31: LLDP_GreenPeriodBegin.Valid
This optional field shall be set according to the Table 176.

Table 176 – LLDP_GreenPeriodBegin.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	The value of LLDP_GreenPeriodBegin.Offset is not valid. It shall be set to zero.
0x01	Valid	The value of LLDP_GreenPeriodBegin.Offset is valid.

4.9.2.16 Coding of the field LLDP_LengthOfPeriod

A port is divided into different time periods. The duration of all periods is shown by this field. The coding of this field shall be according to 3.7.3.5 and to Figure 13. The individual bits shall have the following meaning:

Bit 0 – 30: LLDP_LengthOfPeriod.Length
This field shall be set according to the Table 177.

Table 177 – LLDP_LengthOfPeriod.Length

Value (hexadecimal)	Meaning	Usage
0x00007A12 – 0x003D0900	Duration of a cycle in nanoseconds	The value shall be a multiply of 31 250 ns. See 6.2.4.62
0x003D0900 – 0x7FFFFFFF	Reserved	

Bit 31: LLDP_LengthOfPeriod.Valid
This optional field shall be set according to the Table 178.

Table 178 – LLDP_LengthOfPeriod.Valid

Value (hexadecimal)	Meaning	Usage
0x00	Invalid	LLDP_LengthOfPeriod.Length is not valid. It shall be set to zero.
0x01	Valid	LLDP_LengthOfPeriod.Length is valid.

4.9.2.17 Coding of the field LLDP_POINT

This field shall be coded as data type OctetString with one octet. The value shall be “.”.

NOTE The field LLDP_POINT is not terminated by zero.

4.9.2.18 Coding of the field LLDP_TimeToLive

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.19 Coding of the field LLDP_TLVHeader

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.20 Coding of the field LLDP_ChassisIDSubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.21 Coding of the field LLDP_PortIDSubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.22 Coding of the field LLDPOption

This field shall be coded according to IEEE 802.1AB-2005, 9.4.

4.9.2.23 Coding of the field LLDP_OUI

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.24 Coding of the field LLDP_8021_SubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.25 Coding section related to LLDP_Management

4.9.2.25.1 Coding of the field LLDP_ManagementData

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.26 Coding section related to LLDP_8023

4.9.2.26.1 Coding of the field LLDP_8023_SubType

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.26.2 Coding of the field LLDP_8023_AUTONEG

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.26.3 Coding of the field LLDP_8023_PMDCAP

This field shall be coded according to IEEE 802.1AB-2005.

4.9.2.26.4 Coding of the field LLDP_8023_OPMAU

This field shall be coded according to IEEE 802.1AB-2005.

4.10 MAC bridges

4.10.1 Overview

The concepts according to IEEE 802.1D standard shall be applied. This part of the specification defines the utilization of the IEEE 802.1D standard. It includes extensions for RT_CLASS_3 phase regarding forwarding of frames. The routing mechanisms of IEEE 802.1D shall be temporary disabled within the RT_CLASS_3 phase. In this case, the routing of RT_CLASS_3 frames shall be done according to the attributes defined by the appropriate ASE. The values of the attributes define a special routing table, which is used.

Apart from IEEE 802.1D behaviour, this protocol specification defines the following protocol machines to provide special forwarding actions:

- RT_CLASS_3 Forwarding Protocol Machine (IFW)
- Time Synchronization with follow up frame Forwarding Protocol Machine (S_FU_FW)
- Time Synchronization without follow up frame Forwarding Protocol Machine (S_ FW).

4.10.2 RT_CLASS_3 Forwarding Protocol Machine (IFW)

4.10.2.1 Primitive definitions

The service primitives including their associated parameters issued by IFW user received by IFW and vice versa are described in the MAC bridges ASE in the service definition.

Furthermore, as interface between IFW user and media access control machines common data queues (see 4.16.1) shall be used.

4.10.2.2 IFW state table

The FW state table is shown in Table 179.

Table 179 – IFW state table

#	Current State	Event /Condition =>Action	Next State
1	P_ON	=>	NORMAL
2	NORMAL	/Port != LOCAL && FW_C2_List(Port).Num_entry <> 0 && FW_C2_List(Port).First_entry.DA in FDB => for n := all Ports in FDB_Entry(FW_C2_List(Port).First_entry.DA) PUT_C2_REQ (n) FW_C2_List(Port).Num_entry-- FW_C2_List(Port).Remove()	NORMAL
3	NORMAL	/Port != LOCAL && FW_C2_List(Port).Num_entry <> 0 && !(FW_C2_List(Port).First_entry.DA in FDB) => for n := all Ports \ Port PUT_C2_REQ (n) FW_C2_List(Port).Num_entry-- FW_C2_List(Port).Remove()	NORMAL
4	NORMAL	/Port != LOCAL && FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry <> 0 => SYNC_FW_Req (Port)	NORMAL
5	NORMAL	/Port != LOCAL && FW_C2_List(Port).Num_entry ==0 && FW_S_List(Port).Num_entry == 0 && FW_N_List(Port).Num_entry <> 0 && FW_N_List(Port).First_entry.DA in FDB => for n := all Ports in FDB_Entry(FW_N_List(Port).First_entry.DA) PUT_N_REQ (n) FW_N_List(Port).Num_entry-- FW_N_List(Port).Remove()	NORMAL
6	NORMAL	/Port != LOCAL && FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry == 0 && FW_N_List(Port).Num_entry<>0 && !(FW_N_List(Port).First_entry.DA in FDB) => for n := all Ports \ Port PUT_N_REQ (n) FW_N_List(Port).Num_entry-- FW_N_List(Port).Remove()	NORMAL
7	NORMAL	/Port == LOCAL && FW_C2_List(Port).Num_entry <> 0 && FW_C2_List(Port).First_entry.DA in FDB => for n := all Ports in FDB_Entry(FW_C2_List(Port).First_entry.DA) PUT_C2_REQ (n) SETUP_C2_CNF (Port, OK) FW_C2_List(Port).Num_entry-- FW_C2_List(Port).Remove()	NORMAL

#	Current State	Event /Condition =>Action	Next State
8	NORMAL	/Port == LOCAL && FW_C2_List(Port).Num_entry<>0 &&!(FW_C2_List(Port).First_entry.DA in FDB) => for n := all Ports \ Port PUT_C2_REQ (n) SETUP_C2_CNF (Port, OK) FW_C2_List(Port).Num_entry-- FW_C2_List(Port).Remove()	NORMAL
9	NORMAL	/Port == LOCAL && FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry <> 0 => SYNC_FW_Req (Port)	NORMAL
10	NORMAL	/Port == LOCAL && FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry == 0 && FW_N_List(Port).Num_entry <> 0 && FW_N_List(Port).First_entry.D_Port == AUTO && FW_N_List(Port).First_entry.DA in FDB => for n := all Ports in FDB_Entry(FW_N_List(Port).First_entry.DA) PUT_N_REQ (n) SETUP_N_CNF (Port, OK, NIL) FW_N_List(Port).Num_entry-- FW_N_List(Port).Remove()	NORMAL
11	NORMAL	/Port == LOCAL && FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry == 0 && FW_N_List(Port).Num_entry <> 0 && FW_N_List(Port).First_entry.D_Port == AUTO &&!(FW_N_List(Port).First_entry.DA in FDB) => for n := all Ports \ Port PUT_N_REQ (n) SETUP_N_CNF (Port, OK, NIL) FW_N_List(Port).Num_entry-- FW_N_List(Port).Remove()	NORMAL
12	NORMAL	/Port == LOCAL && FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry == 0 && FW_N_List(Port).Num_entry <> 0 && FW_N_List(Port).First_entry.D_Port != AUTO => PUT_N_REQ (D_Port) FW_N_List(Port).Num_entry-- FW_N_List(Port).Remove()	NORMAL
13	NORMAL	/FW_C2_List(Port).Num_entry == 0 && FW_S_List(Port).Num_entry == 0 && FW_N_List(Port).Num_entry == 0 => ignore	NORMAL

#	Current State	Event /Condition =>Action	Next State
14	NORMAL	/CNF_List(Port).Num_entry <> 0 && CNF_List(Port).First_entry.Function == C2 => GET_CNF (Port)	NORMAL
15	NORMAL	/CNF_List(Port).Num_entry <> 0 && CNF_List(Port).First_entry.Function == C3 => GET_CNF (Port)	NORMAL
16	NORMAL	/CNF_List(Port).Num_entry <> 0 && CNF_List(Port).First_entry.Function == S => SYNC_FW_Cnf (Port)	NORMAL
17	NORMAL	/CNF_List(Port).Num_entry <> 0 && CNF_List(Port).First_entry.Function == N && CNF_List(Port).First_entry.D_Port == AUTO => GET_CNF (Port)	NORMAL
18	NORMAL	/CNF_List(Port).Num_entry <> 0 && CNF_List(Port).First_entry.Function == N && CNF_List(Port).First_entry.D_Port <> AUTO => GET_SETUP_N_CNF (Port)	NORMAL
19	NORMAL	**IFW_Reset_Req ()** => Reset Sched_list for n := all Ports \ Local MMAC_Reset_Ind (n) IFW_Reset_Cnf (OK)	NORMAL
20	NORMAL	**IFW_IRT_Schedule_add_Req (CREP, Port, ReductionRatio, Phase)** => Store in Sched_list IFW_IRT_Schedule_add_Cnf (CREP, Port, OK)	NORMAL
21	NORMAL	**IFW_IRT_Schedule_rem_Req (CREP, Port, ReductionRatio, Phase)** => Delete from Sched_list IFW_IRT_Schedule_rem_Cnf (CREP, Port, OK)	NORMAL
22	NORMAL	**IFW_Schedule_Req (Port, ClockTime, Period, Len)** /Period != Red => MMAC_Set_Period_Ind (Port, Period, Len) IFW_Schedule_Cnf (OK)	NORMAL
23	NORMAL	**IFW_Schedule_Req (Port, ClockTime, Period, Len)** /Period == Red && Sched_List(Port) <> NIL => RedRatio := 1 L_ClockTime := ClockTime MMAC_Set_Period_Ind (Port, Period, Len) IFW_Schedule_Cnf (OK)	CALC_P
24	NORMAL	**IFW_Schedule_Req (Port, ClockTime, Period, Len)** /Period == Red && Sched_List(Port) == NIL => MMAC_Set_Period_Ind (Port, Period, Len) IFW_Schedule_Cnf (OK)	NORMAL

#	Current State	Event /Condition =>Action	Next State
25	CALC_P	/RedRatio <= MAX_REDACTION_RATIO => i := 1 L_Phase := (L_ClockTimer MOD RedRatio) + 1	CHECK_P
26	CALC_P	/RedRatio > MAX_REDACTION_RATIO =>	NORMAL
27	CHECK_P	/Sched_List(Port)(RedRatio)(L_Phase)(i) <> NIL => Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset)	IRTFW
28	CHECK_P	/Sched_List(Port)(RedRatio)(L_Phase)(i) == NIL => RedRatio++	CALC_P
29	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) <> NIL && FW_C3_List(Port).Num_entry <> 0 && FW_C3_List(Port).First_entry.DLSDU.FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && FW_C3_List(Port).First_entry.DLSDU.Len == Sched_List(Port)(RedRatio)(L_Phase)(i).Len => for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) FW_C3_List(Port).Num_entry-- FW_C3_List(Port).Remove() i++ Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset)	IRTFW
30	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) == NIL && FW_C3_List(Port).Num_entry <> 0 && FW_C3_List(Port).First_entry.DLSDU.FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && FW_C3_List(Port).First_entry.DLSDU.Len == Sched_List(Port)(RedRatio)(L_Phase)(i).Len => for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) FW_C3_List(Port).Num_entry-- FW_C3_List(Port).Remove() RedRatio++	CALC_P
31	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) <> NIL && FW_C3_List(Port).Num_entry <> 0 && FW_C3_List(Port).First_entry.DLSDU.FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && FW_C3_List(Port).First_entry.DLSDU.Len <> Sched_List(Port)(RedRatio)(L_Phase)(i).Len => C_SDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len C_SDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := WRONG_LENGTH APDU_Status.TransferStatus := WRONG_LENGTH for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) ---------------------------- C_SDU is undefined FW_C3_List(Port).Num_entry-- FW_C3_List(Port).Remove() i++ Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset) IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)	IRTFW

#	Current State	Event /Condition =>Action	Next State
32	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) == NIL && FW_C3_List(Port).Num_entry<>0 && FW_C3_List(Port).First_entry.DLSDU.FrameID == Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID && FW_C3_List(Port).First_entry.DLSDU.Len <> Sched_List(Port)(RedRatio)(L_Phase)(i).Len => C_SDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len C_SDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := WRONG_LENGTH APDU_Status.TransferStatus := WRONG_LENGTH for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) ---------------------------- C_SDU is undefined FW_C3_List(Port).Num_entry-- FW_C3_List(Port).Remove() RedRatio++ IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)	CALC_P
33	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) <> NIL && FW_C3_List(Port).Num_entry<>0 && FW_C3_List(Port).First_entry.DLSDU.FrameID <> Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID => C_SDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len C_SDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := WRONG_FRAMEID APDU_Status.TransferStatus := WRONG_FRAMEID for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) ---------------------------- C_SDU is undefined FW_C3_List(Port).Num_entry-- FW_C3_List(Port).Remove() i++ Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset) IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)	IRTFW
34	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) == NIL && FW_C3_List(Port).Num_entry<>0 && FW_C3_List(Port).First_entry.DLSDU.FrameID <> Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID => C_SDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len C_SDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := WRONG_FRAMEID APDU_Status.TransferStatus := WRONG_FRAMEID for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) ---------------------------- C_SDU is undefined FW_C3_List(Port).Num_entry-- FW_C3_List(Port).Remove() IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)	CALC_P
35	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) <> NIL && FW_C3_List(Port).Num_entry == 0 => C_SDU.Len := Sched_List(Port)(RedRatio)(L_Phase)(i).Len C_SDU.FrameID := Sched_List(Port)(RedRatio)(L_Phase)(i).FrameID ErrCode := IRT_ERROR APDU_Status.TransferStatus := IRT_ERROR for n := all Ports in Sched_List(Port)(RedRatio)(L_Phase)(i).PortList PUT_C3_REQ (n) ---------------------------- C_SDU is undefined i++ Start IRT_Timer (Sched_List(Port)(RedRatio)(L_Phase)(i).TimeOffset) IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)	IRTFW

#	Current State	Event /Condition =>Action	Next State
36	IRTFW	**IRT_Timer expired** /Sched_List(Port)(RedRatio)(L_Phase)(i+1) == NIL && FW_C3_List(Port).Num_entry == 0 => RedRatio++	CALC_P
37	IRTFW	**IFW_Reset_Req ()** => Reset Sched_list for n := all Ports \ Local MMAC_Reset_Ind (n) IFW_Reset_Cnf (OK)	NORMAL
38	IRTFW	**IFW_IRT_Schedule_add_Req (CREP, Port, ReductionRatio, Phase)** => Store in Sched_list IFW_IRT_Schedule_add_Cnf (CREP, Port, OK)	IRTFW
39	IRTFW	**IFW_IRT_Schedule_rem_Req (CREP, Port, ReductionRatio, Phase)** => Delete from Sched_list IFW_IRT_Schedule_rem_Cnf (CREP, Port, OK)	IRTFW
40	IRTFW	**IFW_Schedule_Req (Port, ClockTime, Period, Len)** => ErrCode := RUNNING_RED_PERIOD IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)	IRTFW

4.10.2.3 Functions

All functions of the IFW are summarized in Table 180.

Table 180 – IFW function table

Function name	Operations
PUT_N_REQ (Port)	Put a service request to the service queue to MMAC. with FW_N_List is selected FW_N_List; N_List of Port N_List.Insert() N_List.Last_Entry.CREP := FW_N_List.First_Entry.CREP N_List.Last_Entry.S_Port := FW_N_List.First_Entry.S_Port N_List.Last_Entry.D_Port := FW_N_List.First_Entry.D_Port N_List.Last_Entry.TStamp := FW_N_List.First_Entry.TStamp N_List.Last_Entry.DA := FW_N_List.First_Entry.DA N_List.Last_Entry.SA := FW_N_List.First_Entry.SA N_List.Last_Entry.VLANPrio := FW_N_List.First_Entry.VLANPrio N_List.Last_Entry.VLANID := FW_N_List.First_Entry.VLANID N_List.Last_Entry.DLSDU := FW_N_List.First_Entry.DLSDU N_List.Last_Entry.Funtion := N N_List.Num_entry++
PUT_C2_REQ (Port)	Put a service request to the service queue to MMAC. with FW_C2_List is selected FW_C2_List; C2_List of Port C2_List.Insert() C2_List.Last_Entry.CREP := FW_C2_List.First_Entry.CREP C2_List.Last_Entry.S_Port := FW_C2_List.First_Entry.S_Port C2_List.Last_Entry.D_Port := FW_C2_List.First_Entry.D_Port C2_List.Last_Entry.TStamp := FW_C2_List.First_Entry.TStamp C2_List.Last_Entry.DA := FW_C2_List.First_Entry.DA C2_List.Last_Entry.SA := FW_C2_List.First_Entry.SA C2_List.Last_Entry.VLANPrio := FW_C2_List.First_Entry.VLANPrio C2_List.Last_Entry.VLANID := FW_C2_List.First_Entry.VLANID C2_List.Last_Entry.DLSDU := FW_C2_List.First_Entry.DLSDU C2_List.Last_Entry.Funtion := C2 C2_List.Num_entry++
PUT_C3_REQ (Port)	Put a service request to the service queue to MMAC. with FW_C3_List is selected FW_C3_List; C3_List of Port C3_List.Insert() C3_List.Last_Entry.CREP := FW_C3_List.First_Entry.CREP C3_List.Last_Entry.S_Port := FW_C3_List.First_Entry.S_Port C3_List.Last_Entry.D_Port := FW_C3_List.First_Entry.D_Port C3_List.Last_Entry.TStamp := FW_C3_List.First_Entry.TStamp C3_List.Last_Entry.DA := FW_C3_List.First_Entry.DA C3_List.Last_Entry.SA := FW_C3_List.First_Entry.SA C3_List.Last_Entry.VLANPrio := FW_C3_List.First_Entry.VLANPrio C3_List.Last_Entry.VLANID := FW_C3_List.First_Entry.VLANID C3_List.Last_Entry.DLSDU := FW_C3_List.First_Entry.DLSDU C3_List.Last_Entry.Funtion := C3 C3_List.Num_entry++
SETUP_N_CNF (Port, Status)	Get a service confirmation from the confirmation queue to FW_N. with FW_N_List is selected FW_N_List CNF_List.Insert() CNF_List.Last_Entry := FW_N_List.First_Entry CNF_List.Last_Entry.Status := Status CNF_List.Last_Entry.TStamp := NIL CNF_List.Last_Entry.Function := N CNF_List.Num_entry++

Function name	Operations
SETUP_C2_CNF (Port, Status)	Get a service confirmation from the confirmation queue to FW_C2. with FW_C2_List is selected FW_C2_List CNF_List.Insert() CNF_List.Last_Entry := FW_C2_List.First_Entry CNF_List.Last_Entry.Status := Status CNF_List.Last_Entry.TStamp := NIL CNF_List.Last_Entry.Function := C2 CNF_List.Num_entry++
SETUP_C3_CNF (Port, Status)	Get a service confirmation from the confirmation queue to FW_C3. with FW_C3_List is selected FW_C3_List CNF_List.Insert() CNF_List.Last_Entry := FW_C3_List.First_Entry CNF_List.Last_Entry.Status := Status CNF_List.Last_Entry.TStamp := NIL CNF_List.Last_Entry.Function := C3 CNF_List.Num_entry++
GET_CNF (Port)	Get a service confirmation from the confirmation queue of MMAC. CNF_List.Num_entry-- CNF_List.Remove()
GET_SETUP_N_CNF (Port)	Get a service confirmation from the confirmation queue of MMAC. CNF_List.Num_entry-- Entry := CNF_List.First_Entry Status := CNF_List.First_Entry.Status TStamp := CNF_List.First_Entry.TStamp CNF_List.Remove() Put a confirmation to the confirmation queue of LMPM. CNF_List.Insert() CNF_List.Last_Entry := Entry CNF_List.Last_Entry.Status := Status CNF_List.Last_Entry.TStamp := TStamp CNF_List.Last_Entry.Function := N CNF_List.Num_entry++

4.10.3 S_FW

4.10.3.1 Primitive definitions

The service primitives including their associated parameters issued by S_FW user received by S_FW and vice versa are described in the MAC bridges ASE in the service definition.

Furthermore, as interface between S_FW user and media access control machines common data queues (see 4.16.1) shall be used.

4.10.3.2 S_FW state table

The S_FW state table is shown in Table 181.

Table 181 – S_FW state table

#	Current State	Event /Condition =>Action	Next State
1	P-ON	=> Ini FWSM_List	IDLE
2	IDLE	=> ReceiptTimer.start (SYNC_RCV_TIMEOUT)	READY
3	READY	**ReceiptTimer.expired ()** => AGING_OF_FWSM_LIST ReceiptTimer.start (SYNC_RCV_TIMEOUT)	READY
4	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry == 0 => ignore	READY
5	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (!CHECK_S_PAR_SYNC_!FU || !CHECK_S_PAR_SYNC_FU || !CHECK_FOLLOWUP_FRAME) => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
6	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && !CHECK_SM_IN_FWSM_LIST && FREE_ENTRY_IN_FWSM_LIST && CHECK_LINE_DELAY => PUT_SM_IN_FWSM_LIST for i := all op. Ports \ Port PUT_S_REQ (Port) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
7	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && !CHECK_SM_IN_FWSM_LIST && FREE_ENTRY_IN_FWSM_LIST && !CHECK_LINE_DELAY => PUT_SM_IN_FWSM_LIST PUT_S_REQ (LOCAL) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
8	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && !CHECK_SM_IN_FWSM_LIST && !FREE_ENTRY_IN_FWSM_LIST => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY

<table>
<tr><th>#</th><th>Current State</th><th>Event
/Condition
=>Action</th><th>Next State</th></tr>
<tr><td>9</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& (CHECK_S_PAR_SYNC_!FU
|| CHECK_S_PAR_SYNC_FU)
&& CHECK_SM_IN_FWSM_LIST
&& !CHECK_SEQ_SYNC
=>
FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
<tr><td>10</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& (CHECK_S_PAR_SYNC_!FU
|| CHECK_S_PAR_SYNC_FU)
&& CHECK_SM_IN_FWSM_LIST
&& CHECK_SEQ_SYNC
&& !CHECK_LINE_DELAY
=>
UPDATE_SM_IN_FWSM_LIST
PUT_S_REQ (LOCAL)

FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
<tr><td>11</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& (CHECK_S_PAR_SYNC_!FU
|| CHECK_S_PAR_SYNC_FU)
&& CHECK_SM_IN_FWSM_LIST
&& CHECK_SEQ_SYNC
&& CHECK_LINE_DELAY
=>
UPDATE_SM_IN_FWSM_LIST
for i := all op. Ports \ Port
  PUT_S_REQ (Port)

FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
<tr><td>12</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port == LOCAL
&& FW_S_List(Port).Num_entry == 0
=>
ignore</td><td>READY</td></tr>
<tr><td>13</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port == LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& !CHECK_S_PAR_SYNC_FU
=>
SETUP_S_CNF (Port, ERRO_PAR)
FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
<tr><td>14</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port == LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& CHECK_S_PAR_SYNC_!FU
&& !CHECK_SM_IN_FWSM_LIST
&& FREE_ENTRY_IN_FWSM_LIST
=>
PUT_SM_IN_FWSM_LIST
for i := all op. Ports \ Port
  PUT_S_REQ (Port)

SETUP_S_CNF (Port, OK)
FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
</table>

#	Current State	Event /Condition =>Action	Next State
15	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_!FU && !CHECK_SM_IN_FWSM_LIST && !FREE_ENTRY_IN_FWSM_LIST => SETUP_S_CNF (Port, FWSM_LIST_FULL) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
16	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC && CHECK_SM_IN_FWSM_LIST && !CHECK_SEQ_SYNC => SETUP_S_CNF (Port, WRONG_SEQ) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
17	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC && CHECK_SM_IN_FWSM_LIST && CHECK_SEQ_SYNC => UPDATE_SM_IN_FWSM_LIST for i := all op. Ports \ Port PUT_S_REQ (Port) SETUP_S_CNF (Port, OK) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
18	READY	**SYNC_FW_CNF (Port)** => CNF_List(Port).Num_entry-- CNF_List(Port).Remove()	READY
19	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && !CHECK_SM_IN_FWSM_LIST => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
20	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && CHECK_SM_IN_FWSM_LIST && !CHECK_FOLLOWUP_RECEIVE => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
21	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && CHECK_SM_IN_FWSM_LIST && CHECK_FOLLOWUP_RECEIVE && !CHECK_PORT_FOLLOWUP => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY

#	Current State	Event /Condition =>Action	Next State
22	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && CHECK_SM_IN_FWSM_LIST && CHECK_PORT_FOLLOWUP && CHECK_SEQ_FOLLOW_UP => UPDATE_FOLLOWUP_IN_FWSM_LIST for i := all op. Ports \ Port PUT_FU_REQ (Port) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY

4.10.3.3 Functions

All functions of the S_FW are summarized in Table 182.

Table 182 – S_FW function table

Function name	Operations
CHECK_S_PAR_SYNC	Check PTCP sync frame LT == RT ((FW_S_List(Port).First_entry.S_SDU.FrameID >= 0x00 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0x20) || (FW_S_List(Port).First_entry.S_SDU.FrameID >= 0xFF00 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0xFF20)) S_SDU.Data: according to DLSDU.Len
CHECK_S_PAR_SYNC_FU	Check PTCP sync frame with followup frame LT == RT FW_S_List(Port).First_entry.S_SDU.FrameID >= 0x20 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0x40 S_SDU.Data: according to DLSDU.Len
CHECK_SM_IN_FWSM_LIST	Check source address of PTCP sync master in sync master list sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) {sm_sa, SyncID} in FWSM_List
FREE_ENTRY_IN_FWSM_LIST	Check for free entry in sync master list FWSM_List.Num_entry < FWSM_MAX_CNT_ENTRY
CHECK_LINE_DELAY	Check line delay on receive port available FW_S_List(Port).LineDelay_Status == OK
CHECK_SEQ_SYNC	sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.seq > FW_S_List(Port).First_entry.S_SDU. PTCP_SequenceID

Function name	Operations
PUT_SM_IN_FWSM_LIST	Store sync master paramter in sync master list sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Insert(sm_sa, SyncID) FWSM_List.Entry{sm_sa, SyncID}.FrameID := FW_S_List(Port).First_entry.S_SDU.FrameID FWSM_List.Entry{sm_sa, SyncID}.SM_SA := FW_S_List(Port).First_entry.S_SDU. PTCP_MasterSourceAddress FWSM_List.Entry{sm_sa, SyncID}.seq := FW_S_List(Port).First_entry.S_SDU. PTCP_SequenceID FWSM_List.Entry{sm_sa, SyncID}.RcvPort := Port FWSM_List.Entry{sm_sa, SyncID}.FollowUp := FALSE FWSM_List.Entry{sm_sa, SyncID}.Receipt := 1 FWSM_List.Num_entry++
UPDATE_SM_IN_FWSM_LIST	Update entry of sync master list sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{sm_sa, SyncID}.seq := FW_S_List(Port).First_entry.S_SDU. PTCP_SequenceID FWSM_List.Entry{sm_sa, SyncID}.RcvPort := Port FWSM_List.Entry.{sm_sa, SyncID}.FollowUp := FALSE FWSM_List.Entry.Receipt++

AGING_OF_FWSM_LIST	Aging of entries in forwarding sync master list
	for m := 0 to FWSM_List.Num_entry
	if (FWSM_List.Entry(m).Receipt == 0)
	SyncID := SYNC_ID_MASK (FWSM_List.Entry(m).FrameID)
	sm_sa :=FWSM_List.Entry(m).SM_SA
	FWSM_List.Num_entry--
	FWSM_List.Remove (sm_sa, SyncID)
	else
	FWSM_List.Entry[m].Receipt := 0

CHECK_FOLLOWUP_FRAME	Check PTCP followup frame of sync frame LT == RT FW_S_List(Port).First_entry.S_SDU.FrameID >= 0xFF20 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0xFF40 S_SDU.Data: according to DLSDU.Len
CHECK_FOLLOWUP_RECEIVE	Check followup frame for sync frames already received sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.FollowUp == FALSE
CHECK_PORT_FOLLOWUP	Check receive port of followup frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.RcvPort == Port
CHECK_SEQ_FOLLOWUP	Check sequence number of followup frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.seq == FW_S_List(Port).First_entry.S_SDU. PTCP_SequenceID

Function name	Operations
UPDATE_FOLLOWUP_IN_FWSM_LIST	Update entry of forwarding sync master list for followup frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{sm_sa, SyncID}.FollowUp := TRUE
PUT_FU_REQ (Port)	Put a service request to the service queue to MMAC. with FW_S_List is selected N_List.Insert() N_List.Last_Entry := FW_S_List.First_Entry N_List.Last_Entry.SA := TS N_List.Last_Entry.Funtion := S N_List.Num_entry++
PUT_S_REQ (Port)	Put a service request to the service queue to MMAC and LMPM. with FW_S_List is selected N_List.Insert() N_List.Last_Entry := FW_S_List.First_Entry N_List.Last_Entry.SA := TS N_List.Last_Entry.Funtion := S N_List.Num_entry++
SETUP_S_CNF (Port, Status)	Get a service confirmation from the confirmation queue to FW_N. with FW_N_List is selected CNF_List.Insert() CNF_List.Last_Entry := FW_S_List.First_Entry CNF_List.Last_Entry.Status := Status CNF_List.Last_Entry.Function := S CNF_List.Num_entry++

4.10.4 S_FU_FW

4.10.4.1 Primitive definitions

The service primitives including their associated parameters issued by S_FU_FW user received by S_FU_FW and vice versa are described in the MAC bridges ASE in the service definition.

Furthermore, as interface between S_FU_FW user and media access control machines common data queues (see 4.16.1) shall be used.

4.10.4.2 S_FU_FW state table

The S_FU_FW state table is shown in Table 183.

Table 183 – S_FU_FW state table

<table>
<tr><th>#</th><th>Current State</th><th>Event
/Condition
=>Action</th><th>Next State</th></tr>
<tr><td>1</td><td>P-ON</td><td>=>
Ini FWSM_List</td><td>IDLE</td></tr>
<tr><td>2</td><td>IDLE</td><td>=>
ReceiptTimer.start (SYNC_RCV_TIMEOUT)</td><td>READY</td></tr>
<tr><td>3</td><td>READY</td><td>ReceiptTimer.expired ()
=>
AGING_OF_FWSM_LIST
ReceiptTimer.start (SYNC_RCV_TIMEOUT)</td><td>READY</td></tr>
<tr><td>4</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry == 0
=></td><td>READY</td></tr>
<tr><td>5</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& (!CHECK_S_PAR_SYNC_!FU
|| !CHECK_S_PAR_SYNC_FU
|| !CHECK_FOLLOWUP_FRAME)
=>
FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
<tr><td>6</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& (CHECK_S_PAR_SYNC_!FU
|| CHECK_S_PAR_SYNC_FU)
&& !CHECK_SM_IN_FWSM_LIST
&& FREE_ENTRY_IN_FWSM_LIST
&& CHECK_LINE_DELAY
=>
PUT_SM_IN_FWSM_LIST
for i := all op. Ports \ Port
  PUT_SYNC_REQ (Port)

FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
<tr><td>7</td><td>READY</td><td>SYNC_FW_Req (Port)
/Port <> LOCAL
&& FW_S_List(Port).Num_entry <> 0
&& (CHECK_S_PAR_SYNC_!FU
|| CHECK_S_PAR_SYNC_FU)
&& !CHECK_SM_IN_FWSM_LIST
&& FREE_ENTRY_IN_FWSM_LIST
&& CHECK_LINE_DELAY
=>
PUT_SM_IN_FWSM_LIST
for i := all op. Ports \ Port
 PUT_SYNC_REQ (Port)

FW_S_List(Port).Num_entry--
FW_S_List(Port).Remove()</td><td>READY</td></tr>
</table>

#	Current State	Event /Condition =>Action	Next State
8	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && !CHECK_SM_IN_FWSM_LIST && FREE_ENTRY_IN_FWSM_LIST && !CHECK_LINE_DELAY => PUT_SM_IN_FWSM_LIST PUT_SYNC_REQ (LOCAL) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
9	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && !CHECK_SM_IN_FWSM_LIST && !FREE_ENTRY_IN_FWSM_LIST => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
10	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && CHECK_SM_IN_FWSM_LIST && !CHECK_SEQ_SYNC => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
11	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && (CHECK_S_PAR_SYNC_!FU || CHECK_S_PAR_SYNC_FU) && CHECK_SM_IN_FWSM_LIST && CHECK_SEQ_SYNC && !CHECK_LINE_DELAY => UPDATE_SM_IN_FWSM_LIST PUT_SYNC_REQ (LOCAL) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
12	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_!FU && CHECK_SM_IN_FWSM_LIST && CHECK_SEQ_SYNC && CHECK_LINE_DELAY => UPDATE_SM_IN_FWSM_LIST for i := all op. Ports \ Port PUT_SYNC_REQ (Port) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY

#	Current State	Event /Condition =>Action	Next State
13	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_FU && CHECK_SM_IN_FWSM_LIST && CHECK_SEQ_SYNC && CHECK_LINE_DELAY => UPDATE_SM_IN_FWSM_LIST (Port) for i := all op. Ports \ Port PUT_SYNC_REQ (Port) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
14	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry == 0 => ignore	READY
15	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && !CHECK_S_PAR_SYNC_!FU => SETUP_S_CNF (Port, ERRO_PAR) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
16	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_!FU && !CHECK_SM_IN_FWSM_LIST && FREE_ENTRY_IN_FWSM_LIST => PUT_SM_IN_FWSM_LIST for i := all op. Ports \ Port PUT_SYNC_REQ (Port) SETUP_S_CNF (Port, OK) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
17	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_!FU && !CHECK_SM_IN_FWSM_LIST && !FREE_ENTRY_IN_FWSM_LIST => SETUP_S_CNF (Port, FWSM_LIST_FULL) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
18	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_!FU && CHECK_SM_IN_FWSM_LIST && !CHECK_SEQ_SYNC => SETUP_S_CNF (Port, WRONG_SEQ) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY

#	Current State	Event /Condition =>Action	Next State
19	READY	**SYNC_FW_Req (Port)** /Port == LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_S_PAR_SYNC_!FU && CHECK_SM_IN_FWSM_LIST && CHECK_SEQ_SYNC => UPDATE_SM_IN_FWSM_LIST for i := all op. Ports \ Port PUT_SYNC_REQ (Port) SETUP_S_CNF (Port, OK) FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
20	READY	**SYNC_REQ_Cnf (Port)** /!CHECK_CNF_SYNC (Port) => CNF_List.Num_entry-- CNF_List.Remove()	
21	READY	**SYNC_REQ_Cnf (Port)** /CHECK_CNF_SYNC (Port) && !CHECK_SYNC_WITH_FU => GET_CNF (Port) CALC_SYNC_DELAY (Port) PUT_NEW_FU_REQ (Port)	
22	READY	**SYNC_REQ_Cnf (Port)** /CHECK_CNF_SYNC (Port) && CHECK_SYNC_WITH_FU => GET_CNF (Port)	
23	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && !CHECK_SM_IN_FWSM_LIST => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
24	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && CHECK_SM_IN_FWSM_LIST && !CHECK_FOLLOWUP_RECEIVE => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY
25	READY	**SYNC_FW_Req (Port)** /Port <> LOCAL && FW_S_List(Port).Num_entry <> 0 && CHECK_FOLLOWUP_FRAME && CHECK_SM_IN_FWSM_LIST && CHECK_FOLLOWUP_RECEIVE && !CHECK_PORT_FOLLOWUP => FW_S_List(Port).Num_entry-- FW_S_List(Port).Remove()	READY

#	Current State	Event /Condition =>Action	Next State

26	READY	**SYNC_FW_Req (Port)**
		/Port <> LOCAL
		&& FW_S_List(Port).Num_entry <> 0
		&& CHECK_FOLLOWUP_FRAME
		&& CHECK_SM_IN_FWSM_LIST
		&& CHECK_PORT_FOLLOWUP
		&& CHECK_SEQ_FOLLOW_UP
		=>
		UPDATE_FOLLOWUP_IN_FWSM_LIST
		for i := all op. Ports \ Port
		{
		if (Port == LOCAL)
		{
		CALC_SYNC_DELAY (LOCAL)
		PUT_FU_REQ (LOCAL)
		}
		if ((Port <> LOCAL) && (CHECK_CNF (Port)))
		{
		CALC_SYNC_DELAY (Port)
		PUT_FU_REQ (Port)
		}
		}
		
		FW_S_List(Port).Num_entry--
		FW_S_List(Port).Remove()
			READY

4.10.4.3 Functions

All functions of the S_FW are summarized in Table 184.

Table 184 – S_FU_FW function table

Function name	Operations
CHECK_S_PAR_SYNC_!FU	Check PTCP sync frame without followup frame LT == RT (FW_S_List(Port).First_entry.S_SDU.FrameID >= 0x0000 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0x0020) || (FW_S_List(Port).First_entry.S_SDU.FrameID >= 0xFF00 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0xFF20) S_SDU.Data: according to DLSDU.Len
CHECK_S_PAR_SYNC_FU	Check PTCP sync frame with followup frame LT == RT FW_S_List(Port).First_entry.S_SDU.FrameID >= 0x20 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0x40 S_SDU.Data: according to DLSDU.Len
CHECK_SM_IN_FWSM_LIST	Check source address of PTCP sync master in sync master list sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) {sm_sa, SyncID} in FWSM_List
FREE_ENTRY_IN_FWSM_LIST	Check for free entry in sync master list FWSM_List.Num_entry < FWSM_MAX_CNT_ENTRY
CHECK_LINE_DELAY	Check line delay on receive port available FW_S_List(Port).LineDelay_Status == OK

Function name	Operations
CHECK_SEQ_SYNC	Check sequence number of sync frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.seq > FW_S_List(Port).First_entry.S_SDU.PTCP_SequneceID
PUT_SM_IN_FWSM_LIST	Store sync master paramter in sync master list sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Insert(sm_sa, SyncID) FWSM_List.Entry{sm_sa, SyncID}.FrameID := FW_S_List(Port).First_entry.S_SDU.FrameID FWSM_List.Entry{sm_sa, SyncID}.SM_SA := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress FWSM_List.Entry{sm_sa, SyncID}.seq := FW_S_List(Port).First_entry.S_SDU.PTCP_SequneceID FWSM_List.Entry{sm_sa, SyncID}.RcvPort := Port FWSM_List.Entry{sm_sa, SyncID}.Rcv_T2 := FW_S_List(Port).First_entry.TStamp FWSM_List.Entry{sm_sa, SyncID}.Snd_T1[all Ports] := 0 FWSM_List.Entry{sm_sa, SyncID}.Conf[all Ports] := FALSE FWSM_List.Entry{sm_sa, SyncID}.FollowUp := FALSE FWSM_List.Entry{sm_sa, SyncID}.Receipt := 1 FWSM_List.Num_entry++
UPDATE_SM_IN_FWSM_LIST	Update entry of sync master list sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{sm_sa, SyncID}.FrameID := FW_S_List(Port).First_entry.S_SDU.FrameID FWSM_List.Entry{sm_sa, SyncID}.seq := FW_S_List(Port).First_entry.S_SDU.PTCP_SequneceID FWSM_List.Entry{sm_sa, SyncID}.RcvPort := Port FWSM_List.Entry{sm_sa, SyncID}.Rcv_T2 := FW_S_List(Port).First_entry.TStamp FWSM_List.Entry{sm_sa, SyncID}.Snd_T1[all Ports] := 0 FWSM_List.Entry{sm_sa, SyncID}.Cnf[all Ports] := FALSE FWSM_List.Entry{sm_sa, SyncID}.FollowUp := FALSE FWSM_List.Entry.Receipt++

AGING_OF_FWSM_LIST	Aging of entries in forwarding sync master list
	for m := 0 to FWSM_List.Num_entry
	  if (FWSM_List.Entry(m).Receipt == 0)
	    SyncID := SYNC_ID_MASK (FWSM_List.Entry(m).FrameID)
	    sm_sa :=FWSM_List.Entry(m).SM_SA
	    FWSM_List.Num_entry--
	    FWSM_List.Remove (sm_sa, SyncID)
	  else
	    FWSM_List.Entry[m].Receipt := 0

Function name	Operations
CHECK_CNF_SYNC (Port)	Check confirmation of sync request LT == RT sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) CNF_List(Port).First_entry.S_SDU.FrameID >= 0x20 && CNF_List(Port).First_entry.S_SDU.FrameID < 0x40 FWSM_List.Entry{sm_sa, SyncID}.Cnf[Port] == FALSE FWSM_List.Entry{sn_sa, SyncID).seq == CNF_List.First_entry.S_SDU.PTCP_SequenceID
GET_CNF (Port)	Get a service confirmation from the confirmation queue of MMAC. CNF_List.Num_entry-- sm_sa := CNF_List.First_Entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (CNF_List.First_Entry.S_SDU.FrameID) Entry := CNF_List.First_Entry Status := CNF_List.First_Entry.Status FWSM_List.Entry{sm_sa, SyncID}.Snd_T1[Port] := CNF_List.First_Entry.TStamp FWSM_List.Entry{sm_sa, SyncID}.Cnf[Port] := TRUE CNF_List.Remove()
CHECK_CNF (Port)	Check confirmation of sync request already received sm_sa := FW_N_List.First_Entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_N_List.First_Entry.S_SDU.FrameID) FWSM_List.Entry{sm_sa, SyncID}.Cnf[Port] := TRUE

CALC_SYNC_DELAY (Port)	Calculate residental time of Sync Frame
	if (Port == LOCAL)
	SyncDelay := LineDelay (Entry.S_Port) * CORRECTION (SYNC_ID_MASK (Entry.S_SDU.FrameID))
	else
	Delay := FWSM_List.Entry{sm_sa, SyncID}.Snd_T1[Port] - FWSM_List.Entry{sm_sa, SyncID}.Rcv_T2)
	SyncDelay := (Delay + LineDelay (Entry.S_Port)) * CORRECTION (SYNC_ID_MASK (Entry.S_SDU.FrameID))

PUT_NEW_FU_REQ (Port)	Set FollowUp-Flag in FWSM_List FWSM_List.Entry{sm_sa, SyncID}.FollowUp := TRUE Put a service request to the service queue to MMAC N_List.Insert() N_List.Last_Entry.CREP := Entry.CREP N_List.Last_Entry.S_Port := Port N_List.Last_Entry.TStamp := NIL N_List.Last_Entry.DA := FOLLOWUP_MULTICAST (Entry.S_SDU.FrameID) N_List.Last_Entry.SA := TS N_List.Last_Entry.Prio := Entry.Prio N_List.Last_Entry.VLAN_Tag := Entry.VLAN_Tag N_List.Last_Entry.LT := RT N_List.Last_Entry.S_SDU := Entry.S_SDU N_List.Last_Entry.S_SDU.Delay := SyncDelay N_List.Last_Entry.S_SDU.FrameID := FOLLOWUP_RT_ID (Entry.S_SDU.FrameID) N_List.Num_entry++

Function name	Operations
PUT_FU_REQ (Port)	Put a service request to the service queue to MMAC. with FW_S_List is selected N_List.Insert() N_List.Last_Entry := FW_S_List.First_Entry N_List.Last_Entry.SA := TS N_List.Last_Entry.S_SDU.Delay := FW_S_List.First_Entry.S_SDU.SyncDelay + SyncDelay N_List.Last_Entry.Funtion := S N_List.Num_entry++
CHECK_FOLLOWUP_FRAME	Check PTCP followup frame of sync frame LT == RT FW_S_List(Port).First_entry.S_SDU.FrameID >= 0xFF20 && FW_S_List(Port).First_entry.S_SDU.FrameID < 0xFF40 S_SDU.Data: according to DLSDU.Len
CHECK_FOLLOWUP_RECEIVE	Check followup frame for sync frames already received sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.FollowUp == FALSE
CHECK_PORT_FOLLOWUP	Check receive port of followup frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.RcvPort == Port
CHECK_SEQ_FOLLOWUP	Check sequence number of followup frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.seq == FW_S_List(Port).First_entry.S_SDU.PTCP_SequneceID
UPDATE_FOLLOWUP_IN_FWSM_LIST	Update entry of forwarding sync master list for followup frame sm_sa := FW_S_List(Port).First_entry.S_SDU.PTCP_MasterSourceAddress SyncID:= SYNC_ID_MASK (FW_S_List(Port).First_entry.S_SDU.FrameID) FWSM_List.Entry{ sm_sa, SyncID}.FollowUp := TRUE
PUT_SYNC_REQ (Port)	Put a service request to the service queue to MMAC and LMPM. with FW_S_List is selected N_List.Insert() N_List.Last_Entry := FW_S_List.First_Entry N_List.Last_Entry.SA := TS N_List.Last_Entry.S_SDU.FrameID := FW_S_List.First_Entry.S_SDU.FrameID | FLAG_FOLLOWUP N_List.Last_Entry.Funtion := S N_List.Num_entry++
SETUP_S_CNF (Port, Status)	Get a service confirmation from the confirmation queue to FW_N. with FW_N_List is selected CNF_List.Insert() CNF_List.Last_Entry := FW_S_List.First_Entry CNF_List.Last_Entry.Status := Status CNF_List.Last_Entry.TStamp := NIL CNF_List.Last_Entry.Function := S CNF_List.Num_entry++

4.11 Virtual bridges

4.11.1 Overview

The concepts according to IEEE 802.1Q standard shall be applied. This part of the specification defines the utilization of the IEEE 802.1Q standard. It includes extensions for RT_CLASS_3 phase regarding forwarding of frames. The routing mechanisms of IEEE 802.1Q shall be temporary disabled within the RT_CLASS_3 phase. In this case, the routing of RT_CLASS_3 frames shall be done according to the attributes defined by the appropriate ASE. The values of the attributes define a special routing table, which is used.

Apart from IEEE 802.1Q standard behaviour, this protocol specification defines the following protocol machines to provide special MAC actions:

- Mapping MAC Protocol Machine (MMAC)

4.11.2 MMAC

4.11.2.1 Primitive definitions

4.11.2.1.1 Primitives exchanged between MMAC user and MMAC

Table 185 shows the primitives issued by the MMAC user to the MMAC.

Table 185 – Primitives issued by LMPM to MMAC

Primitive Name	Associated Parameters
MMAC_RESET.req	None

Table 186 shows the primitives issued by the MMAC to the MMAC user.

Table 186 – Primitives issued by MMAC to LMPM

Primitive Name	Associated Parameters
MMAC_RESET.cnf	none

4.11.2.1.2 Primitives exchanged between MMAC and MAC

Table 187 shows the service primitives including their associated parameters issued by the MMAC and received by the MAC.

Table 187 – Primitives issued by MMAC to MAC

Primitive name	Source	Associated parameters	Functions
M_UNITDATA.req	MMAC	frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, rif_information (optional), include_tag	

Table 188 shows the service primitives including their associated parameters issued by the MAC and received by the MMAC.

Table 188 – Primitives issued by MAC to MMAC

Primitive name	Source	Associated parameters	Functions
M_UNITDATA.ind	MAC	frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier, rif_information (optional)	
M_UNITDATA.cnf	MAC		

4.11.2.1.3 Parameters of DMPM primitives

A data request primitive is invoked in order to generate a M_UNITDATA.request primitive, as defined in the Internal Sublayer Service, IEEE 802.3.

The frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority and frame_check_sequence parameters are as defined for the M_UNITDATA.req primitive of the Internal Sublayer service.

The definition of the canonical_format_indicator parameter is as defined for the E-ISS data indication. The vlan_classification parameter carries the VLAN classification assigned to the frame by the Ingress rules. The rif_information parameter, if present, carries the value of any RIF information to be associated with the request. The include_tag parameter carries a Boolean value. True indicates to the service provider that the mac_service_data_unit parameter of the data request shall include a Tag Header. False indicates that a Tag Header shall not be included. The frame_type, mac_action, destination_address, source_address and frame_check_sequence parameters carry values equal to the corresponding parameters in the received data indication.

4.11.2.2 State machine description

The MMAC is responsible for mapping LMPM services to MMAC.

The other main task is the correct execution of services that are passed to and returned to LMPM via queues.

Local variables of the MMAC

Port_State
This local variable contains the state of the port.

destination_address
This local variable contains the destination MAC address.

Cy_List
This local variable contains the cyclic service send list.

Ac_List
This local variable contains the acyclic service send list.

ListOrgAdd
This local variable contains the list of destination MAC addresses, e.g. PTCP_MulticastMACAdd, LLDP_MulticastMACAdd, MRPMulticastMACAdd and MRRTMulticastMACAdd, which passes ports not in state Forwarding or Disabled.

N_List
This local variable contains non realtime service send list.

4.11.2.3 MMAC state table

The MMAC state table is shown in Table 189.

Table 189 – MMAC state table

<table>
<tr><th>#</th><th>Current State</th><th>Event
/Condition
=>Action</th><th>Next State</th></tr>
<tr><td>1</td><td>P_ON</td><td>=>
Interval := Green</td><td>IDLE</td></tr>
<tr><td>2</td><td>IDLE</td><td>/C2_List.Num_entry <> 0
&& Port_State != Forward
&& !(C2_List.First_Entry.DA in ListOrgAdd)
=>
SETUP_CNF (PORT_BLOCKED)</td><td>IDLE</td></tr>
<tr><td>3</td><td>IDLE</td><td>/N_List.Num_entry <> 0
&& Port_State != Forward
&& !(N_List.First_Entry.DA in ListOrgAdd)
=>
SETUP_CNF (PORT_BLOCKED)</td><td>IDLE</td></tr>
<tr><td>4</td><td>IDLE</td><td>/Interval == Green
&& C2_list.Num_entry <> 0
&& (Port_State == Forward
|| (C2_List.First_Entry.DA in ListOrgAdd))

=>
SETUP_C2_REQ
M_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
<tr><td>5</td><td>IDLE</td><td>/Interval == Green
&& C2_list.Num_entry == 0
&& N_List.Num_entry <> 0
&& (Port_State == Forward
|| (C2_List.First_Entry.DA in ListOrgAdd))

=>
SETUP_N_REQ
M_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
<tr><td>6</td><td>IDLE</td><td>/Interval == Green
&& C2_list.Num_entry == 0
&& N_List.Num_entry == 0
=>
ignore</td><td>IDLE</td></tr>
<tr><td>7</td><td>IDLE</td><td>/Interval == Yellow && C2_list.Num_entry <> 0 && (Port_State == Forward|| (C2_List.First_Entry.DA in ListOrgAdd)) && CalcTx(C2_List.First_Entry.N_SDU.len) =< (Start_Red-TCC.cv)=>SETUP_C2_REQM_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
</table>

<table>
<tr><th>#</th><th>Current State</th><th>Event
/Condition
=>Action</th><th>Next State</th></tr>
<tr><td>8</td><td>IDLE</td><td>/Interval == Yellow
&& C2_list.Num_entry <> 0
&& (Port_State == Forward
|| (C2_List.First_Entry.DA in ListOrgAdd)
|| (N_List.First_Entry.DA in ListOrgAdd))
&& CalcTx(C2_List.First_Entry.N_SDU.len) > (Start_Red-TCC.cv)
&& N_List.Num_entry <> 0
&& CalcTx(N_List.First_Entry.N_SDU.len) =< (Start_Red-TCC.cv)
=>
SETUP_N_REQ
M_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
<tr><td>9</td><td>IDLE</td><td>/Interval == Yellow
&& C2_list.Num_entry == 0
&& N_List.Num_entry <> 0
&& (Port_State == Forward
|| (N_List.First_Entry.DA in ListOrgAdd))
&& CalcTx(N_List.First_Entry.N_SDU.len) =< (Start_Red-TCC.cv)
=>
SETUP_N_REQ
M_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
<tr><td>10</td><td>IDLE</td><td>/Interval == Yellow
&& C2_list.Num_entry == 0
&& N_List.Num_entry <> 0
&& (Port_State == Forward
|| (destination_address in ListOrgAdd))
&& CalcTx(N_List.First_Entry.N_SDU.len) > (Start_Red-TCC.cv)
=>
ignore</td><td>IDLE</td></tr>
<tr><td>11</td><td>IDLE</td><td>/Interval == Yellow && C2_list.Num_entry == 0 && N_List.Num_entry == 0
=>ignore</td><td>IDLE</td></tr>
<tr><td>12</td><td>IDLE</td><td>/Interval == ORANGE
&& C2_list.Num_entry <> 0
&& (Port_State == Forward
|| (C2_List.First_Entry.DA in ListOrgAdd))

=>
SETUP_C2_REQ
M_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
<tr><td>13</td><td>IDLE</td><td>/Interval == ORANGE
&& C2_list.Num_entry == 0
=>
ignore</td><td>IDLE</td></tr>
<tr><td>14</td><td>IDLE</td><td>/Interval == Red
&& C3_list.Num_entry <> 0
=>
SETUP_C3_REQ
M_UNITDATA.req (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, access_priority, frame_check_sequence, canonical_format_indicator, vlan_classification, include_tag)</td><td>AW-CNF</td></tr>
</table>

#	Current State	Event /Condition =>Action	Next State
15	IDLE	/Interval == Red && C3_List.Num_entry == 0 => ignore	IDLE
16	AW-CNF	**M_UNITDATA.cnf (Status)** => SETUP_CNF (Status)	IDLE
17	IDLE / AW-CNF	**MMAC_Reset_Ind (Port)** => RESET_ALL_LIST Interval := GREEN	IDLE
18	IDLE / AW-CNF	**MMAC_Set_Period_Ind(Port, Period, Len)** /Period == Green => Interval := Green	IDLE / AW-CNF
19	IDLE / AW-CNF	**MMAC_Set_Period_Ind(Port, Period, Len)** /Period == Yellow => Interval := Yellow Start_Red := TCC.cv + Len	IDLE / AW-CNF
20	IDLE / AW-CNF	**MMAC_Set_Period_Ind(Port, Period, Len)** /Period == Orange => Interval := Orange	IDLE / AW-CNF
21	IDLE / AW-CNF	**MMAC_Set_Period_Ind(Port, Period, Len)** /Period == Red => Interval := Red	IDLE / AW-CNF
22	IDLE / AW-CNF	**M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)** /Port_State != Forward && MRRT_Mode == FALSE && !(destination_address in ListOrgAdd) && mac_service_data_unit.LT != RT && ((mac_service_data_unit.LT == RT) && (mac_service_data_unit.rt_id < 0x7F || mac_service_data_unit.rt_id > 0x8000)) => ignore	IDLE / AW-CNF
23	IDLE / AW-CNF	**M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)** /mac_service_data_unit.LT == RT && (mac_service_data_unit.rt_id > 0x7F && mac_service_data_unit.rt_id < 0x8000)) => SETUP_C3_IND (Port)	IDLE / AW-CNF
24	IDLE / AW-CNF	**M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)** /(Port_State == Forward || MRRT_Mode == TRUE || (destination_address in ListOrgAdd)) && mac_service_data_unit.LT == RT && mac_service_data_unit.rt_id > 0x7FFF && mac_service_data_unit.rt_id < 0xC000 => SETUP_C2_IND (Port)	IDLE / AW-CNF

#	Current State	Event /Condition =>Action	Next State
25	IDLE / AW-CNF	**M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)** /(Port_State == Forward || (destination_address in ListOrgAdd)) && mac_service_data_unit.LT == RT && ((mac_service_data_unit.rt_id >= 0x00 && mac_service_data_unit.rt_id < 0x40) || (mac_service_data_unit.rt_id >= 0xFF00 && mac_service_data_unit.rt_id < 0xFF40) => SETUP_S_IND (Port)	IDLE / AW-CNF
26	IDLE / AW-CNF	**M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)** /(Port_State == Forward || (destination_address in ListOrgAdd)) && ((mac_service_data_unit.LT == RT && mac_service_data_unit.rt_id >= 0x40 && mac_service_data_unit.rt_id < 0x80) || (mac_service_data_unit.LT == RT && mac_service_data_unit.rt_id >= 0xC000 && mac_service_data_unit.rt_id < 0xFF00) || (mac_service_data_unit.LT == RT && mac_service_data_unit.rt_id >= 0xFF40 && mac_service_data_unit.rt_id <= 0xFFFF)) || mac_service_data_unit.LT <> RT => SETUP_N_IND (Port)	IDLE / AW-CNF
27	IDLE / AW-CNF	**M_UNITDATA.ind (Port, frame_type, mac_action, destination_address, source_address, mac_service_data_unit, user_priority, frame_check_sequence, canonical_format_indicator, vlan_identifier)** /(Port_State == Forward || (destination_address in ListOrgAdd)) && mac_service_data_unit.LT != RT => SETUP_N_IND (Port)	IDLE / AW-CNF

4.11.2.4 Functions

All functions of the MMAC are summarized in Table 190.

Table 190 – MMAC function table

Function name	Operations
SETUP_N_REQ	mac_service_data_unit := N_List.First_Entry.DLSDU source_address := N_List.First_Entry.SA destination_address := N_List.First_Entry.DA user_priority := N_List.First_Entry.VLANPrio access_priority := N_List.First_Entry.VLANPrio canonical_format_indicator := 1 vlan_classification := N_List.First_Entry.VLANID calculate frame_check_sequence Service := N
SETUP_C2_REQ	mac_service_data_unit := C2_List.First_Entry.DLSDU source_address := C2_List.First_Entry.SA destination_address := C2_List.First_Entry.DA user_priority := C2_List.First_Entry.VLANPrio access_priority := C2_List.First_Entry.VLANPrio canonical_format_indicator := 1 vlan_classification := C2_List.First_Entry.VLANID calculate frame_check_sequence Service := C2
SETUP_C3_REQ	mac_service_data_unit := C2_List.First_Entry.DLSDU source_address := C3_List.First_Entry.SA destination_address := C3_List.First_Entry.DA user_priority := C3_List.First_Entry.VLANPrio access_priority := C3_List.First_Entry.VLANPrio canonical_format_indicator := 1 vlan_classification := C3_List.First_Entry.VLANID calculate frame_check_sequence Service := C3
SETUP_N_IND (Port)	FW_N_List.Insert() FW_N_List.Last_Entry.Function := N FW_N_List.First_Entry.CREP := NIL FW_N_List.Last_Entry.S_Port := Port FW_N_List.Last_Entry.D_Port := NIL FW_N_List.Last_Entry.TStamp := GetRcvTimeStamp (Port) FW_N_List.Last_Entry.DA := destination address FW_N_List.Last_Entry.SA := source address FW_N_List.Last_Entry.VLANPrio := user_priority FW_N_List.Last_Entry.VLANID := vlan_identifier FW_N_List.Last_Entry.LT := frame_type FW_N_List.Last_Entry.DLSDU := mac_service_data_unit FW_N_List.Num_entry++
SETUP_S_IND (Port)	FW_S_List.Insert() FW_S_List.Last_Entry.Function := S FW_S_List.First_Entry.CREP := NIL FW_S_List.Last_Entry.S_Port := Port FW_S_List.Last_Entry.D_Port := NIL FW_S_List.Last_Entry.TStamp := GetRcvTimeStamp (Port) FW_S_List.Last_Entry.LD_Status := GetLineDelayStatus (Port) FW_S_List.Last_Entry.DA := destination address FW_S_List.Last_Entry.SA := source address FW_S_List.Last_Entry.VLANPrio := user_priority FW_S_List.Last_Entry.VLANID := vlan_identifier FW_S_List.Last_Entry.LT := frame_type FW_S_List.Last_Entry.DLSDU := mac_service_data_unit FW_S_List.Num_entry++

Function name	Operations
SETUP_C2_IND (Port)	FW_C2_List.Insert() FW_C2_List.Last_Entry.Function := C2 FW_C2_List.First_Entry.CREP := NIL FW_C2_List.Last_Entry.S_Port := Port FW_C2_List.Last_Entry.D_Port := NIL FW_C2_List.Last_Entry.TStamp := GetRcvTimeStamp (Port) FW_C2_List.Last_Entry.DA := destination address FW_C2_List.Last_Entry.SA := source address FW_C2_List.Last_Entry.VLANPrio := user_priority FW_C2_List.Last_Entry.VLANID := vlan_identifier FW_C2_List.Last_Entry.LT := frame_type FW_C2_List.Last_Entry.DLSDU := mac_service_data_unit FW_C2_List.Num_entry++
SETUP_C3_IND (Port)	FW_C3_List.Insert() FW_C3_List.Last_Entry.Function := C3 FW_C3_List.First_Entry.CREP := NIL FW_C3_List.Last_Entry.S_Port := Port FW_C3_List.Last_Entry.D_Port := NIL FW_C3_List.Last_Entry.TStamp := GetRcvTimeStamp (Port) FW_C3_List.Last_Entry.DA := destination address FW_C3_List.Last_Entry.SA := source address FW_C3_List.Last_Entry.VLANPrio := user_priority FW_C3_List.Last_Entry.VLANID := vlan_identifier FW_C3_List.Last_Entry.LT := frame_type FW_C3_List.Last_Entry.DLSDU := mac_service_data_unit FW_C3_List.Num_entry++
SETUP_CNF (Status)	With Service as List–Prefix …_List.Num_entry-- CNF_List.Insert() CNF_List.Last_Entry := …_List.First_Entry CNF_List.Last_Entry.TStamp := GetSndTimeStamp (Port) CNF_List.Last_Entry.Status := Status CNF_List.Num_entry++ …_List.Remove()

RESET_ALL_LIST	While (N_List.Num_entry ≠ 0) SETUP_CNF (RESET)
	While (C2_List.Num_entry ≠ 0) SETUP_CNF (RESET)
	While (C3_List.Num_entry ≠ 0) SETUP_CNF (RESET)

MRRT_Mode	TRUE if bumpless media redundancy is activated

4.12 IP suite

4.12.1 Overview

The utilization of the RFC 768 (UDP), RFC 791 (IP), RFC 792 (ICMP), RFC 826 (ARP), RFC 1112 (IP Multicasting), RFC 2474 (IP Extensions) standards are defined for this specification. It includes definitions for UDP ports and IP multicast addresses and utilization of IP header fields for RT_CLASS_UDP.

4.12.2 IP/UDP syntax description

4.12.2.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

4.12.2.2 IP/UDP APDU abstract syntax

Table 191 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 191 shall represent the content of the DLSDU in Table 4.

Table 191 – IP/UDP APDU syntax

APDU name	APDU structure
ICMP-PDU	IPHeader [a], ICMPHeader, Data*
[a] The value of IPHeader.IP_Protocol shall be set to 1 to indicate ICMP.	

Table 192 defines structures for substitutions of elements of the APDU structures shown in Table 191.

Table 192 – IP/UDP substitutions

Substitution name	Structure
IPHeader	IP_VersionIHL(0x45), IP_TypeOfService, IP_TotalLength, IP_Identification, IP_Flags_FragOffset [a], IP_TTL, IP_Protocol, IP_HeaderChecksum, IP_SrcIPAddress, IP_DstIPAddress, [IP_Options] [b] The encoding of the fields shall be according to RFC 791.
ICMPHeader	ICMP_Type, ICMP_Code, ICMP_HeaderChecksum
UDPHeader	UDP_SrcPort, UDP_DstPort, UDP_DataLength, UDP_Checksum [c]
[a] For UDP-RTC-PDU and UDP-RTA-PDU fragmentation shall not be used. [b] The field IP_Options shall be omitted for UDP-RTC-PDU and UDP-RTA-PDU. [c] The UDP_Checksum should be set to zero for RT_CLASS_UDP and RTA_CLASS_UDP to improve performance. It is not required for the receiver to check the UDP_Checksum for UDP-RTC-PDU and UDP-RTA-PDU.	

4.12.3 IP/UDP transfer syntax

4.12.3.1 Coding section related to IP, ICMP, and UDP

4.12.3.1.1 Coding section related to ICMP

4.12.3.1.1.1 Coding of the field ICMP_Type

The encoding of the fields shall be according to RFC 792.

4.12.3.1.1.2 Coding of the field ICMP_Code

The encoding of the fields shall be according to RFC 792.

4.12.3.1.1.3 Coding of the field ICMP_HeaderChecksum

The encoding of the fields shall be according to RFC 792.

4.12.3.1.2 Coding section related to UDP

4.12.3.1.2.1 Coding of the field UDP_DataLength

The encoding of the fields shall be according to RFC 768.

4.12.3.1.2.2 Coding of the field UDP_Checksum

The encoding of the fields shall be according to RFC 768.

4.12.3.1.2.3 Coding of the field UDP_SrcPort

The encoding of the fields shall be according to RFC 768. Defined values are shown in Table 193.

Table 193 – UDP_SrcPort

Value (hexadecimal)	Meaning	Usage
0x8892	IANA_PNIO_UDP_UNICAST_PORT	UDP-RTC-PDU and UDP-RTA-PDU
0x8893	IANA_PNIO_UDP_MULTICAST_PORT	Reserved
0x8894	IANA_PNIO_EPM_PORT	NDREPMapLookupReq or NDREPMapLookupFreeReq

4.12.3.1.2.4 Coding of the field UDP_DstPort

The encoding of the fields shall be according to RFC 768. Defined values are shown in Table 194.

Table 194 – UDP_DstPort

Value (hexadecimal)	Meaning	Usage
0x8892	IANA_PNIO_UDP_UNICAST_PORT	UDP-RTC-PDU and UDP-RTA-PDU
0x8893	IANA_PNIO_UDP_MULTICAST_PORT	Reserved
0x8894	IANA_PNIO_EPM_PORT	NDREPMapLookupReq or NDREPMapLookupFreeReq, also possible for each RPC call

4.12.3.1.3 Coding section related to IP

4.12.3.1.3.1 Coding of the field IP_DstIPAddress

The encoding of the fields shall be according to RFC 791, RFC 1112, and RFC 2365. Defined values ar shown in Table 195 and Table 196.

Table 195 – IP_DstIPAddress

IP address	Range	Meaning
224.0.0.34	subnet	SUBNET_MULTICAST_IP_ADDRESS_FOR_DISCOVERY
tbd	site	SITE_MULTICAST_IP_ADDRESS_FOR_DISCOVERY

Table 196 – IP Multicast DstIPAddress according to RFC 2365

Multicast IP address	Associated Multicast MAC address	Associated FrameID	Usage
239.0.0.0 – 239.192.247.255			not used
239.192.248.0	01-00-5E-40-F8-00	0xF800	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.248.1 – 239.192.251.254	01-00-5E-40-F8-01 – 01-00-5E-40-FB-FE	0xF801 – 0xFBFE	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.192.251.255	01-00-5E-40-FB-FF	0xFBFF	Used for multicast communication relations in conjunction with RT_CLASS_UDP
239.193.252.0 – 239.255.255.255			not used

4.12.3.1.3.2 Coding of the field IP_VersionIHL

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.3 Coding of the field IP_TypeOfService

The encoding of the fields shall be according to RFC 791 and RFC 2474.

4.12.3.1.3.4 Coding of the field IP_TotalLength

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.5 Coding of the field IP_Identification

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.6 Coding of the field IP_Flags_FragOffset

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.7 Coding of the field IP_TTL

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.8 Coding of the field IP_Protocol

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.9 Coding of the field IP_HeaderChecksum

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.10 Coding of the field IP_SrcIPAddress

The encoding of the fields shall be according to RFC 791.

4.12.3.1.3.11 Coding of the field IP_Options

The encoding of the fields shall be according to RFC 791.

4.13 Domain name system

The utilization of the RFC 1034 (DNS) standard is defined for this specification. It includes the usage and the content syntax of domain name.

4.14 Dynamic host configuration

The utilization of the RFC 2131 (DHCP) standard is defined for this specification. It includes the usage and the content syntax of Client ID.

4.15 Simple network management

4.15.1 Overview

The utilization of the RFC 2674 (Bridges with traffic classes), RFC 2737 (MIB 2), RFC 2863 (IF MIB), RFC 3418 (SNMP), RFC 3621 (Power over Ethernet MIB), and RFC 3636 (MAU MIB) standards are defined for this specification. It includes the usage of different MIBs. Furthermore, a LLDP EXT MIB is specified in 4.15.3 and the PNIO MIB in 4.15.4.

4.15.2 Enterprise number

Coding of the field enterprise number as PNIO MIB identifier. The value shall be according to the Table 197.

Table 197 – Enterprise number

Value (decimal)	Meaning
24 686	Enterprise number for PNIO MIB

4.15.3 LLDP EXT MIB

```
LLDP-EXT-PNO-MIB DEFINITIONS ::= BEGIN

IMPORTS
    SnmpAdminString                FROM SNMP-FRAMEWORK-MIB
```

```
     Unsigned32,
     MODULE-IDENTITY,
     OBJECT-TYPE                         FROM SNMPv2-SMI
     MacAddress,
     DisplayString,
     TruthValue                          FROM SNMPv2-TC
     MODULE-COMPLIANCE,
     OBJECT-GROUP                    FROM SNMPv2-CONF
     lldpExtensions,
     lldpRemTimeMark,
     lldpRemLocalPortNum,
     lldpRemIndex,
     lldpPortConfigEntry,
     lldpLocPortNum                      FROM LLDP-MIB;

lldpXPnoMIB                              MODULE-IDENTITY
     LAST-UPDATED                    "200603090000Z" -- March 9, 2006
     ORGANIZATION                    "PROFIBUS International (PNO)"
     CONTACT-INFO                        "
                                         URL:http://www.profibus.com
                                         email:   info@profibus.com

                                         Postal:  Haid-und-Neu-Strasse 7
                                                  D-76131 Karlsruhe

                                         Tel:++49 721 9658 - 590
                                         "
     DESCRIPTION                         "
                                         The LLDP Management Information Base extension module for
                                     PROFINET organizationally defined discovery information.

                                     Copyright (C) PROFIBUS Nutzerorganisation e.V. (2005).
                                         "
     REVISION                        "200612300000Z" -- December 30, 2006
     DESCRIPTION                         "initial version"
-- OUI for PNO 3791 (00-0E-CF)
     ::= { lldpExtensions 3791 }

------------------------------------------------------------------------------
--
-- Organizationally Defined Information Extension - PNO
--
------------------------------------------------------------------------------

lldpXPnoObjects                          OBJECT IDENTIFIER ::= { lldpXPnoMIB 1 }

-- LLDP PNO extension MIB groups
lldpXPnoConfig                           OBJECT IDENTIFIER ::= { lldpXPnoObjects 1 }
lldpXPnoLocalData                        OBJECT IDENTIFIER ::= { lldpXPnoObjects 2 }
lldpXPnoRemoteData                       OBJECT IDENTIFIER ::= { lldpXPnoObjects 3 }

------------------------------------------------------------------------------
-- PNO - Configuration
------------------------------------------------------------------------------

lldpXPnoConfigTable                      OBJECT-TYPE
     SYNTAX                          SEQUENCE OF LldpXPnoConfigEntry
     MAX-ACCESS                          not-accessible
     STATUS                              current
     DESCRIPTION                         "
                                       A table that controls selection of LLDP
                                       TLVs to be transmitted on individual ports.
                                       "
     ::= { lldpXPnoConfig 1 }

lldpXPnoConfigEntry                      OBJECT-TYPE
     SYNTAX                              LldpXPnoConfigEntry
     MAX-ACCESS                          not-accessible
     STATUS                              current
     DESCRIPTION                         "
                                       LLDP configuration information that controls the
                                       transmission of PNO organizationally defined TLVs
                                       on LLDP transmission capable ports.

                                       This configuration object augments the
                                       lldpPortConfigEntry of the LLDP-MIB, therefore it
                                       is only present along with the port configuration
                                       defined by the associated lldpPortConfigEntry entry.

                                       Each active lldpXPnoConfigEntry must be restored
                                       from non-volatile storage (along with the corresponding
                                       lldpPortConfigEntry) after a re-initialization of the
                                       management system.
                                         "
     AUGMENTS                        { lldpPortConfigEntry}
     ::= { lldpXPnoConfigTable 1 }

LldpXPnoConfigEntry ::=                  SEQUENCE {
     lldpXPnoConfigSPDTxEnable       TruthValue,
     lldpXPnoConfigPortStatusTxEnable TruthValue,
     lldpXPnoConfigAliasTxEnable         TruthValue,
     lldpXPnoConfigMrpTxEnable       TruthValue,
     lldpXPnoConfigPtcpTxEnable          TruthValue
}

lldpXPnoConfigSPDTxEnable            OBJECT-TYPE
     SYNTAX                              TruthValue
     MAX-ACCESS                          read-write
     STATUS                              current
     DESCRIPTION                         "
                                       The lldpXPnoConfigSPDTxEnable, which is defined as
                                       a truth value and configured by the network management,
                                       determines whether the PNO organizationally defined
                                       signal propagation delay TLV transmission is allowed on
                                       a given LLDP transmission capable port.
```

```
                                The signal propagation delay is composed of the port
                                transmission delay, the port receiving delay and the line
                                delay. These values can't be transmitted independently of
                                each other.

                                The value of this object must be restored from non-volatile
                                storage after a re-initialization of the management system.
                                  "
    DEFVAL                        { true }
    ::= { lldpXPnoConfigEntry 1 }

lldpXPnoConfigPortStatusTxEnable  OBJECT-TYPE
    SYNTAX                        TruthValue
    MAX-ACCESS                    read-write
    STATUS                        current
    DESCRIPTION                   "
                                The lldpXPnoConfigPortStatusTxEnable, which is defined as
                                a truth value and configured by the network management,
                                determines whether the PNO organizationally defined
                                RT port status TLV transmission is allowed on a given
                                LLDP transmission capable port.

                                The value of this object must be restored from non-volatile
                                storage after a re-initialization of the management system.
                                  "
    DEFVAL                        { true }
    ::= { lldpXPnoConfigEntry 2 }

lldpXPnoConfigAliasTxEnable         OBJECT-TYPE
    SYNTAX                        TruthValue
    MAX-ACCESS                    read-write
    STATUS                        current
    DESCRIPTION                   "
                                The lldpXPnoConfigAliasTxEnable, which is defined as
                                a truth value and configured by the network management,
                                determines whether the PNO organizationally defined
                                alias TLV (chassisId) transmission is allowed on a given
                                LLDP transmission capable port.

                                The value of this object must be restored from non-volatile
                                storage after a re-initialization of the management system.
                                  "
    DEFVAL                        { true }
    ::= { lldpXPnoConfigEntry 3 }

lldpXPnoConfigMrpTxEnable         OBJECT-TYPE
    SYNTAX                        TruthValue
    MAX-ACCESS                    read-write
    STATUS                        current
    DESCRIPTION                   "
                                The lldpXPnoConfigMrpTxEnable, which is defined as
                                a truth value and configured by the network management,
                                determines whether the PNO organizationally defined
                                MRP TLV transmission is allowed on a given
                                LLDP transmission capable port.

                                The value of this object must be restored from non-volatile
                                storage after a re-initialization of the management system.
                                  "
    DEFVAL                        { true }
    ::= { lldpXPnoConfigEntry 4 }

lldpXPnoConfigPtcpTxEnable          OBJECT-TYPE
    SYNTAX                        TruthValue
    MAX-ACCESS                    read-write
    STATUS                        current
    DESCRIPTION                   "
                                The lldpXPnoConfigPtcpTxEnable, which is defined as
                                a truth value and configured by the network management,
                                determines whether the PNO organizationally defined
                                PTCP TLV transmission is allowed on a given
                                LLDP transmission capable port.

                                The value of this object must be restored from non-volatile
                                storage after a re-initialization of the management system.
                                  "
    DEFVAL                        { true }
    ::= { lldpXPnoConfigEntry 5 }

------------------------------------------------------------------------------
-- PNO - Local System Information
------------------------------------------------------------------------------

lldpXPnoLocTable                  OBJECT-TYPE
    SYNTAX                        SEQUENCE OF LldpXPnoLocEntry
    MAX-ACCESS                    not-accessible
    STATUS                        current
    DESCRIPTION                   "
                                This table contains one row per port for PNO
                                organizationally defined LLDP extension on the
                                local system known to this agent.
                                "
    ::= { lldpXPnoLocalData 1 }

lldpXPnoLocEntry                  OBJECT-TYPE
    SYNTAX                        LldpXPnoLocEntry
    MAX-ACCESS                    not-accessible
    STATUS                        current
    DESCRIPTION                   "
                                Additional information about a particular port
                                component.

                                This object is indexed by the lldpLocPortNum
                                of the LLDP-MIB, therefore it is only present
```

```
                                 along with the port entry defined by the
                                 associated lldpLocPortEntry entry.

                                 Each active lldpXPnoLocEntry must be restored
                                 from non-volatile storage (along with the
                                 corresponding lldpLocPortEntry) after a
                                 re-initialization of the management system.
                                 "
    INDEX                          { lldpLocPortNum   }
    ::= { lldpXPnoLocTable 1 }

LldpXPnoLocEntry ::=             SEQUENCE {
    lldpXPnoLocLPDValue              Unsigned32,
    lldpXPnoLocPortTxDValue        Unsigned32,
    lldpXPnoLocPortRxDValue        Unsigned32,
    lldpXPnoLocPortStatusRT2     INTEGER,
    lldpXPnoLocPortStatusRT3     INTEGER,
    lldpXPnoLocPortNoS                 DisplayString,
    lldpXPnoLocPortMrpUuId             OCTET STRING,
    lldpXPnoLocPortMrrtStatus    INTEGER,
    lldpXPnoLocPortPtcpMaster    MacAddress,
    lldpXPnoLocPortPtcpSubdomainUUID OCTET STRING,
    lldpXPnoLocPortPtcpIRDataUUID OCTET STRING,
    lldpXPnoLocPortModeRT3             INTEGER,
    lldpXPnoLocPortPeriodLength  Unsigned32,
    lldpXPnoLocPortPeriodValidity TruthValue,
    lldpXPnoLocPortRedOffset     Unsigned32,
    lldpXPnoLocPortRedValidity         TruthValue,
    lldpXPnoLocPortOrangeOffset        Unsigned32,
    lldpXPnoLocPortOrangeValidity      TruthValue,
    lldpXPnoLocPortGreenOffset         Unsigned32,
    lldpXPnoLocPortGreenValidity  TruthValue
}

lldpXPnoLocLPDValue                    OBJECT-TYPE
    SYNTAX                       Unsigned32
    UNITS                              "ns"
    MAX-ACCESS                         read-only
    STATUS                             current
    DESCRIPTION                      "
                                 This integer value represents the line propagation
                                 delay in nanoseconds which was measured by the local
                                 system on the corresponding port.

                                 A value of zero shall be used if the system either
                                 could not accomplish the measurement or does not support
                                 such a measurement.
                                 "
    DEFVAL                             { 0 }
    ::= { lldpXPnoLocEntry 1 }

lldpXPnoLocPortTxDValue                OBJECT-TYPE
    SYNTAX                       Unsigned32
    UNITS                              "ns"
    MAX-ACCESS                         read-only
    STATUS                             current
    DESCRIPTION                      "
                                 This integer value represents the PortTxDelay
                                 in nanoseconds which was measured by the local
                                 system on the corresponding port.

                                 A value of zero shall be used if the system either
                                 could not accomplish the measurement or does not
                                 support such a measurement.
                                 "
    DEFVAL                             { 0 }
    ::= { lldpXPnoLocEntry 2 }

lldpXPnoLocPortRxDValue                OBJECT-TYPE
    SYNTAX                       Unsigned32
    UNITS                              "ns"
    MAX-ACCESS                         read-only
    STATUS                             current
    DESCRIPTION                      "
                                 This integer value represents the PortRxDelay
                                 in nanoseconds which was measured by the local
                                 system on the corresponding port.

                                 A value of zero shall be used if the system either
                                 could not accomplish the measurement or does not
                                 support such a measurement.
                                 "
    DEFVAL                             { 0 }
    ::= { lldpXPnoLocEntry 3 }

lldpXPnoLocPortStatusRT2         OBJECT-TYPE
    SYNTAX                       INTEGER {
                                       off(0),
                                       configured(1),
                                       running(2)
                                 }
    MAX-ACCESS                         read-only
    STATUS                             current
    DESCRIPTION                      "
                                 This value represents the status of the corresponding
                                 port of the local system according to RT class 2.

                                 A value of off(0) means that there isn't any RT2
                                 capability available for this port. When the port is
                                 configured for RT2 mode, but the mode isn't active yet
                                 the value will be configured(1). If the RT2 mode is
                                 configured for this port and the mode is active, the
                                 value will be running(2).
                                 "
    ::= { lldpXPnoLocEntry 4 }
```

```
lldpXPnoLocPortStatusRT3          OBJECT-TYPE
    SYNTAX                        INTEGER {
                                      off(0),
                                      configured(1),
                                      up(2),
                                      down(3),
                                      running(4)
                                  }
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    This value represents the status of the corresponding
                                    port of the local system according to RT class 3.

                                    A value of off(0) means that there isn't any RT3
                                    capability available for this port. When the port is
                                    configured for RT3 mode, but the mode isn't active yet
                                    the value will be configured(1).
                                    When the port is ready for transmission and reception
                                    of RT3 traffic, the port status will be running(4).
                                    "
    ::= { lldpXPnoLocEntry 5 }

lldpXPnoLocPortNoS                    OBJECT-TYPE
    SYNTAX                        DisplayString
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    The local PROFINET NameofStation. If the station isn't
                                    configured yet, the value of this object will be the
                                    MAC address of the device as a string.
                                    "
    ::= { lldpXPnoLocEntry 6 }

lldpXPnoLocPortMrpUuId                OBJECT-TYPE
    SYNTAX                        OCTET STRING (SIZE (16))
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    The UUID of the MRP domain to which this port belongs
                                    to. If the port doesn't belong to a MRP domain, the value
                                    must be NIL ('0000000000000000')
                                    "
    ::= { lldpXPnoLocEntry 7 }

lldpXPnoLocPortMrrtStatus         OBJECT-TYPE
    SYNTAX                        INTEGER {
                                      off(0),
                                      configured(1),
                                      up(2)
                                  }
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    This object reports the status of the MRRT entity of the
                                    corresponding port.

                                    A value of off(0) means that there isn't any MRRT
                                    capability available for this port or it is switched off.
                                    The value configured(1) indicates that MRRT is configured
                                    for the port. When MRRT is active on the port, the value
                                    will be up(2).
                                    "
    ::= { lldpXPnoLocEntry 8 }

lldpXPnoLocPortPtcpMaster         OBJECT-TYPE
    SYNTAX                        MacAddress
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    The interface MAC address of the PTCP sync master device
                                    of the PTCP subdomain. If unknown it shall be set to zero.
                                    "
    ::= { lldpXPnoLocEntry 9 }

lldpXPnoLocPortPtcpSubdomainUUID  OBJECT-TYPE
    SYNTAX                        OCTET STRING (SIZE (16))
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    The UUID of the PTCP subdomain to which this port belongs
                                    to. If the port doesn't belong to a PTCP subdomain or the subdomain
                                    is invalid or unknown the value of this object must be
                                    NIL ('0000000000000000').
                                    "
    ::= { lldpXPnoLocEntry 10 }

lldpXPnoLocPortPtcpIRDataUUID     OBJECT-TYPE
    SYNTAX                        OCTET STRING (SIZE (16))
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                    The UUID of the IR data domain to which this port belongs
                                    to. If the port doesn't belong to a IR data domain or the domain
                                    is invalid or unknown the value of this object must be
                                    NIL ('0000000000000000').
                                    "
    ::= { lldpXPnoLocEntry 11 }

lldpXPnoLocPortModeRT3                OBJECT-TYPE
    SYNTAX                        INTEGER
                                  {
                                      standard(1),
                                      optimized(0)
```

```
                                    }
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      The mode in which the RT3 status is given. If the object is
                                      in standard mode all five values of RT3 status are valid, else only
                                      ...
                                      "
    ::= { lldpXPnoLocEntry 12 }

lldpXPnoLocPortPeriodLength             OBJECT-TYPE
    SYNTAX                          Unsigned32 (31250..4000000)
    UNITS                               "ns"
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      This integer value represents the duration of a cycle
                                      in nanoseconds on the corresponding port.
                                      The value shall be a multiply of 31250 nanoseconds.
                                      "
    ::= { lldpXPnoLocEntry 13 }

lldpXPnoLocPortPeriodValidity       OBJECT-TYPE
    SYNTAX                          TruthValue
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      The value of this object indicates whether the value of
                                      lldpXPnoLocPortPeriodLength of the according table entry is
                                      valid or not.
                                      "
    ::= { lldpXPnoLocEntry 14 }

lldpXPnoLocPortRedOffset            OBJECT-TYPE
    SYNTAX                          Unsigned32 (0..3999999)
    UNITS                               "ns"
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      This integer value represents the begin of the RT_CLASS_3 period
                                      of the cycle of the receive direction on the corresponding port
                                      as an offset relative to the begin of the cycle in nanoseconds.
                                      "
    ::= { lldpXPnoLocEntry 15 }

lldpXPnoLocPortRedValidity              OBJECT-TYPE
    SYNTAX                          TruthValue
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      The value of this object indicates whether the value of
                                      lldpXPnoLocPortRedOffset of the according table entry is
                                      valid or not.
                                      "
    ::= { lldpXPnoLocEntry 16 }

lldpXPnoLocPortOrangeOffset             OBJECT-TYPE
    SYNTAX                          Unsigned32 (0..3999999)
    UNITS                               "ns"
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      This integer value represents the begin of the RT_CLASS_2 period
                                      of the cycle of the receive direction on the corresponding port
                                      as an offset relative to the begin of the cycle in nanoseconds.
                                      "
    ::= { lldpXPnoLocEntry 17 }

lldpXPnoLocPortOrangeValidity       OBJECT-TYPE
    SYNTAX                          TruthValue
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      The value of this object indicates whether the value of
                                      lldpXPnoLocPortOrangeOffset of the according table entry is
                                      valid or not.
                                      "
    ::= { lldpXPnoLocEntry 18 }

lldpXPnoLocPortGreenOffset              OBJECT-TYPE
    SYNTAX                          Unsigned32 (0..3999999)
    UNITS                               "ns"
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      This integer value represents the begin of the unrestricted period
                                      of the cycle of the receive direction on the corresponding port
                                      as an offset relative to the begin of the cycle in nanoseconds.
                                      "
    ::= { lldpXPnoLocEntry 19 }

lldpXPnoLocPortGreenValidity        OBJECT-TYPE
    SYNTAX                          TruthValue
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION                       "
                                      The value of this object indicates whether the value of
                                      lldpXPnoLocPortGreenOffset of the according table entry is
                                      valid or not.
                                      "
    ::= { lldpXPnoLocEntry 20 }

------------------------------------------------------------------------------
-- PNO - Remote System Information
------------------------------------------------------------------------------
```

```
lldpXPnoRemTable                  OBJECT-TYPE
    SYNTAX                            SEQUENCE OF LldpXPnoRemEntry
    MAX-ACCESS                        not-accessible
    STATUS                        current
    DESCRIPTION                       "
                                    This table contains one or more rows per physical network
                                    connection known to this agent. The agent may wish to
                                    ensure that only one lldpXPnoRemEntry is present for
                                    each local port, or it may choose to maintain multiple
                                    lldpXPnoRemEntries for the same local port.
                                    "
    ::= { lldpXPnoRemoteData 1 }

lldpXPnoRemEntry                  OBJECT-TYPE
    SYNTAX                        LldpXPnoRemEntry
    MAX-ACCESS                    not-accessible
    STATUS                        current
    DESCRIPTION                       "
                                    Each entry represents the received information of the
                                    communication partner on this physical connection.

                                    The entries feature multiple indices from the
                                    lldpRemEntry of the LLDP-MIB, therefore it is
                                    only present along with the description defined
                                    by the associated lldpRemEntry entry.
                                    "
    INDEX                             { lldpRemTimeMark, lldpRemLocalPortNum, lldpRemIndex }
    ::= { lldpXPnoRemTable 1 }

LldpXPnoRemEntry ::=              SEQUENCE {
    lldpXPnoRemLPDValue               Unsigned32,
    lldpXPnoRemPortTxDValue         Unsigned32,
    lldpXPnoRemPortRxDValue         Unsigned32,
    lldpXPnoRemPortStatusRT2        INTEGER,
    lldpXPnoRemPortStatusRT3        INTEGER,
    lldpXPnoRemPortNoS                DisplayString,
    lldpXPnoRemPortMrpUuId            OCTET STRING,
    lldpXPnoRemPortMrrtStatus     INTEGER,
    lldpXPnoRemPortPtcpMaster     MacAddress,
    lldpXPnoRemPortPtcpSubdomainUUID OCTET STRING,
    lldpXPnoRemPortPtcpIRDataUUID OCTET STRING,
    lldpXPnoRemPortModeRT3            INTEGER,
    lldpXPnoRemPortPeriodLength   Unsigned32,
    lldpXPnoRemPortPeriodValidity TruthValue,
    lldpXPnoRemPortRedOffset      Unsigned32,
    lldpXPnoRemPortRedValidity        TruthValue,
    lldpXPnoRemPortOrangeOffset       Unsigned32,
    lldpXPnoRemPortOrangeValidity     TruthValue,
    lldpXPnoRemPortGreenOffset        Unsigned32,
    lldpXPnoRemPortGreenValidity  TruthValue
}

lldpXPnoRemLPDValue                   OBJECT-TYPE
    SYNTAX                        Unsigned32
    UNITS                             "ns"
    MAX-ACCESS                    read-only
    STATUS                        current
    DESCRIPTION                       "
                                    This integer value represents the line propagation
                                    delay in nanoseconds which was measured by the remote
                                    system on the corresponding port.

                                    A value of zero shall be used if the remote system either
                                    could not accomplish the measurement or does not support
                                    such a measurement.
                                    "
    ::= { lldpXPnoRemEntry 1 }

lldpXPnoRemPortTxDValue               OBJECT-TYPE
    SYNTAX                        Unsigned32
    UNITS                             "ns"
    MAX-ACCESS                    read-only
    STATUS                        current
    DESCRIPTION                       "
                                    This integer value represents the PortTxDelay in
                                    nanoseconds which was measured by the remote
                                    system on the corresponding port.

                                    A value of zero shall be used if the remote system either
                                    could not accomplish the measurement or does not support
                                    such a measurement.
                                    "
    ::= { lldpXPnoRemEntry 2 }

lldpXPnoRemPortRxDValue               OBJECT-TYPE
    SYNTAX                        Unsigned32
    UNITS                             "ns"
    MAX-ACCESS                    read-only
    STATUS                        current
    DESCRIPTION                       "
                                    This integer value represents the PortRxDelay in
                                    nanoseconds which was measured by the remote
                                    system on the corresponding port.

                                    A value of zero shall be used if the remote system either
                                    could not accomplish the measurement or does not support
                                    such a measurement.
                                    "
    ::= { lldpXPnoRemEntry 3 }

lldpXPnoRemPortStatusRT2          OBJECT-TYPE
    SYNTAX                        INTEGER {
                                      off(0),
                                      configured(1),
```

```
                                    running(2)
                                }
    MAX-ACCESS                      read-only
    STATUS                          current
    DESCRIPTION
                                  "
                                  This value represents the status of the corresponding
                                  port of the remote system according to RT class 2.

                                  A value of off(0) means that there isn't any RT2
                                  capability available for this port. When the port is
                                  configured for RT2 mode, but the mode isn't active yet
                                  the value will be configured(1). If the RT2 mode is
                                  configured for this port and the mode is active, the
                                  value will be running(2).
                                  "
    ::= { lldpXPnoRemEntry 4 }

lldpXPnoRemPortStatusRT3            OBJECT-TYPE
    SYNTAX                          INTEGER {
                                        off(0),
                                        configured(1),
                                        up(2),
                                        down(3),
                                        running(4)
                                    }
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION
                                  "
                                  This value represents the status of the corresponding
                                  port of the remote system according to RT class 3.

                                  A value of off(0) means that there isn't any RT3
                                  capability available for this port. When the port is
                                  configured for RT3 mode, but the mode isn't active yet
                                  the value will be configured(1).
                                  When the port is ready for transmission and reception
                                  of RT3 traffic, the port status will be running(4).
                                  "
    ::= { lldpXPnoRemEntry 5 }

lldpXPnoRemPortNoS                      OBJECT-TYPE
    SYNTAX                          DisplayString
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION
                                  "
                                  The PROFINET NameofStation of the remote partner. If the
                                  station isn't configured yet, the value of this object
                                  will be the MAC address of the device as a string.
                                    "
    ::= { lldpXPnoRemEntry 6 }

lldpXPnoRemPortMrpUuId                  OBJECT-TYPE
    SYNTAX                          OCTET STRING (SIZE (16))
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION
                                  "
                                  The UUID of the MRP domain to which the corresponding port
                                  of the remote system belongs to. If the port doesn't belong
                                  to a MRP domain, the value must be NIL ('0000000000000000').
                                  "
    ::= { lldpXPnoRemEntry 7 }

lldpXPnoRemPortMrrtStatus           OBJECT-TYPE
    SYNTAX                          INTEGER {
                                        off(0),
                                        configured(1),
                                        up(2)
                                    }
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION
                                  "
                                  This object reports the status of the MRRT entity of the
                                  corresponding port.

                                  A value of off(0) means that there isn't any MRRT
                                  capability available for this port or it is switched off.
                                  The value configured(1) indicates that MRRT is configured
                                  for the port. When MRRT is active on the port, the value
                                  will be up(2).
                                  "
    ::= { lldpXPnoRemEntry 8 }

lldpXPnoRemPortPtcpMaster           OBJECT-TYPE
    SYNTAX                          MacAddress
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION
                                  "
                                  The interface MAC address of the PTCP sync master device
                                  of the PTCP subdomain. If unknown it shall be set to zero.
                                  "
    ::= { lldpXPnoRemEntry 9 }

lldpXPnoRemPortPtcpSubdomainUUID    OBJECT-TYPE
    SYNTAX                          OCTET STRING (SIZE (16))
    MAX-ACCESS                          read-only
    STATUS                              current
    DESCRIPTION
                                  "
                                  The UUID of the PTCP subdomain to which this port belongs
                                  to. If the port doesn't belong to a PTCP subdomain or the subdomain
                                  is invalid or unknown the value of this object must be
                                  NIL ('0000000000000000').
                                  "
    ::= { lldpXPnoRemEntry 10 }

lldpXPnoRemPortPtcpIRDataUUID       OBJECT-TYPE
```

```
    SYNTAX                     OCTET STRING (SIZE (16))
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 The UUID of the IR data domain to which this port belongs
                                 to. If the port doesn't belong to a IR data domain or the domain
                                 is invalid or unknown the value of this object must be
                                 NIL ('0000000000000000').
                                 "
    ::= { lldpXPnoRemEntry 11 }

lldpXPnoRemPortModeRT3             OBJECT-TYPE
    SYNTAX                     INTEGER
                               {
                                   standard(1),
                                   optimized(0)
                               }
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 The mode in which the RT3 status is given. If the object is
                                 in standard mode all five values of RT3 status are valid, else only
                                 ...
                                 "
    ::= { lldpXPnoRemEntry 12 }


lldpXPnoRemPortPeriodLength        OBJECT-TYPE
    SYNTAX                     Unsigned32 (31250..4000000)
    UNITS                          "ns"
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 This integer value represents the duration of a cycle
                                 in nanoseconds on the corresponding port.
                                 The value shall be a multiply of 31250 nanoseconds.
                                 "
    ::= { lldpXPnoRemEntry 13 }

lldpXPnoRemPortPeriodValidity      OBJECT-TYPE
    SYNTAX                     TruthValue
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 The value of this object indicates whether the value of
                                 lldpXPnoRemPortPeriodLength of the according table entry is
                                 valid or not.
                                 "
    ::= { lldpXPnoRemEntry 14 }

lldpXPnoRemPortRedOffset           OBJECT-TYPE
    SYNTAX                     Unsigned32 (0..3999999)
    UNITS                          "ns"
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 This integer value represents the begin of the RT_CLASS_3 period
                                 of the cycle of the receive direction on the corresponding port
                                 as an offset relative to the begin of the cycle in nanoseconds.
                                 "
    ::= { lldpXPnoRemEntry 15 }

lldpXPnoRemPortRedValidity         OBJECT-TYPE
    SYNTAX                     TruthValue
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 The value of this object indicates whether the value of
                                 lldpXPnoRemPortRedOffset of the according table entry is
                                 valid or not.
                                 "
    ::= { lldpXPnoRemEntry 16 }

lldpXPnoRemPortOrangeOffset        OBJECT-TYPE
    SYNTAX                     Unsigned32 (0..3999999)
    UNITS                          "ns"
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 This integer value represents the begin of the RT_CLASS_2 period
                                 of the cycle of the receive direction on the corresponding port
                                 as an offset relative to the begin of the cycle in nanoseconds.
                                 "
    ::= { lldpXPnoRemEntry 17 }

lldpXPnoRemPortOrangeValidity      OBJECT-TYPE
    SYNTAX                     TruthValue
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 The value of this object indicates whether the value of
                                 lldpXPnoRemPortOrangeOffset of the according table entry is
                                 valid or not.
                                 "
    ::= { lldpXPnoRemEntry 18 }

lldpXPnoRemPortGreenOffset         OBJECT-TYPE
    SYNTAX                     Unsigned32 (0..3999999)
    UNITS                          "ns"
    MAX-ACCESS                     read-only
    STATUS                         current
    DESCRIPTION                  "
                                 This integer value represents the begin of the unrestricted period
                                 of the cycle of the receive direction on the corresponding port
                                 as an offset relative to the begin of the cycle in nanoseconds.
```

```
                                        "
    ::= { lldpXPnoRemEntry 19 }

lldpXPnoRemPortGreenValidity      OBJECT-TYPE
    SYNTAX                        TruthValue
    MAX-ACCESS                        read-only
    STATUS                            current
    DESCRIPTION                     "
                                  The value of this object indicates whether the value of
                                  lldpXPnoRemPortGreenOffset of the according table entry is
                                  valid or not.
                                  "
    ::= { lldpXPnoRemEntry 20 }

-------------------------------------------------------------------------------
-- Conformance Information
-------------------------------------------------------------------------------

lldpXPnoConformance                   OBJECT IDENTIFIER ::= { lldpXPnoMIB 2 }
lldpXPnoCompliances                   OBJECT IDENTIFIER ::= { lldpXPnoConformance 1 }
lldpXPnoGroups                        OBJECT IDENTIFIER ::= { lldpXPnoConformance 2 }

-- compliance statements

lldpXPnoCompliance                    MODULE-COMPLIANCE
    STATUS                            current
    DESCRIPTION                       "
                                  The compliance statement for SNMP entities which
                                  implement the PNO organizationally defined LLDP
                                  extension MIB.
                                  "
    MODULE                            -- compliant to this module
    MANDATORY-GROUPS              { lldpXPnoConfigGroup, lldpXPnoLocGroup, lldpXPnoRemGroup }
    GROUP                             lldpXPnoMrpGroup
    DESCRIPTION                       "Required if the system provides MRP."
    GROUP                             lldpXPnoPtcpGroup
    DESCRIPTION                       "Required if the system provides PTCP."
    ::= { lldpXPnoCompliances 1 }

-- MIB groupings

lldpXPnoConfigGroup                   OBJECT-GROUP
    OBJECTS                           {
                                      lldpXPnoConfigSPDTxEnable,
                                      lldpXPnoConfigPortStatusTxEnable,
                                      lldpXPnoConfigAliasTxEnable
                                      }
    STATUS                            current
    DESCRIPTION                       "
                                  The collection of objects which are used to configure the
                                  PNO organizationally defined LLDP extension implementation behavior.

                                  This group is mandatory for agents which implement the PNO
                                  organizationally defined LLDP extension, because the information
                                  about the signal propagation delay is necessary to configure
                                  PROFINET domains.
                                  "
    ::= { lldpXPnoGroups 1 }

lldpXPnoLocGroup                  OBJECT-GROUP
    OBJECTS                           { lldpXPnoLocLPDValue,
                                  lldpXPnoLocPortTxDValue,
                                  lldpXPnoLocPortRxDValue,
                                  lldpXPnoLocPortStatusRT2,
                                  lldpXPnoLocPortStatusRT3,
                                  lldpXPnoLocPortNoS,
                                  lldpXPnoLocPortModeRT3,
                                  lldpXPnoLocPortPeriodLength,
                                  lldpXPnoLocPortPeriodValidity,
                                  lldpXPnoLocPortRedOffset,
                                  lldpXPnoLocPortRedValidity,
                                  lldpXPnoLocPortOrangeOffset,
                                  lldpXPnoLocPortOrangeValidity,
                                  lldpXPnoLocPortGreenOffset,
                                  lldpXPnoLocPortGreenValidity
                                  }
    STATUS                            current
    DESCRIPTION                       "
                                  The collection of objects which are used to configure the
                                  PNO organizationally defined LLDP extension implementation behavior.

                                  This group is mandatory for agents which implement the PNO
                                  organizationally defined LLDP extension, because the information
                                  about the signal propagation delay is necessary to configure
                                  PROFINET domains.
                                  "
    ::= { lldpXPnoGroups 2 }

lldpXPnoRemGroup                  OBJECT-GROUP
    OBJECTS                           { lldpXPnoRemLPDValue,
                                  lldpXPnoRemPortTxDValue,
                                  lldpXPnoRemPortRxDValue,
                                  lldpXPnoRemPortStatusRT2,
                                  lldpXPnoRemPortStatusRT3,
                                  lldpXPnoRemPortNoS,
                                  lldpXPnoRemPortModeRT3,
                                  lldpXPnoRemPortPeriodLength,
                                  lldpXPnoRemPortPeriodValidity,
                                  lldpXPnoRemPortRedOffset,
                                  lldpXPnoRemPortRedValidity,
                                  lldpXPnoRemPortOrangeOffset,
                                  lldpXPnoRemPortOrangeValidity,
                                  lldpXPnoRemPortGreenOffset,
                                  lldpXPnoRemPortGreenValidity
                                  }
```

```
    STATUS                            current
    DESCRIPTION                       "
                                    The collection of objects which are used to configure the
                                    PNO organizationally defined LLDP extension implementation behavior.

                                    This group is mandatory for agents which implement the PNO
                                    organizationally defined LLDP extension, because the information
                                    about the signal propagation delay is necessary to configure
                                    PROFINET domains.
                                      "
    ::= { lldpXPnoGroups 3 }

lldpXPnoMrpGroup                  OBJECT-GROUP
    OBJECTS                           { lldpXPnoConfigMrpTxEnable,
                                    lldpXPnoLocPortMrpUuId,
                                    lldpXPnoLocPortMrrtStatus,
                                    lldpXPnoRemPortMrpUuId,
                                    lldpXPnoRemPortMrrtStatus
                                  }
    STATUS                            current
    DESCRIPTION                       "
                                    The collection of objects which are used to configure the
                                    PNO organizationally defined LLDP extension implementation behavior.

                                    This group is mandatory for agents which implement the PNO
                                    organizationally defined LLDP extension, because the information
                                    about the signal propagation delay is necessary to configure
                                    PROFINET domains.
                                      "
    ::= { lldpXPnoGroups 4 }

lldpXPnoPtcpGroup                     OBJECT-GROUP
    OBJECTS                           { lldpXPnoConfigPtcpTxEnable,
                                    lldpXPnoLocPortPtcpMaster,
                                    lldpXPnoLocPortPtcpSubdomainUUID,
                                    lldpXPnoLocPortPtcpIRDataUUID,
                                    lldpXPnoRemPortPtcpMaster,
                                    lldpXPnoRemPortPtcpSubdomainUUID,
                                    lldpXPnoRemPortPtcpIRDataUUID
                                  }
    STATUS                            current
    DESCRIPTION                       "
                                    The collection of objects which are used to configure the
                                    PNO organizationally defined LLDP extension implementation behavior.

                                    This group is mandatory for agents which implement the PNO
                                    organizationally defined LLDP extension, because the information
                                    about the signal propagation delay is necessary to configure
                                    PROFINET domains.
                                      "
    ::= { lldpXPnoGroups 5 }

END
```

4.15.4 PNIO MIB

NOTE The PNIO MIB is managed by the PROFIBUS International. It is available at <http://www.profinet.com>.

4.16 Common DLL mapping protocol machines (DMPM)

4.16.1 Overview

The DLL Mapping Protocol consists of several protocol machines:

- Handling of the services invoked by other Application Layer entities (LMPM)
- Forwarding for RT_CLASS_3 and synchronization frames with and without follow-up (IFW, IRT_FW, S_FW),

NOTE Forwarding of standard frames is according to IEEE 802.1D and therefore not specified here.

- Mapping to Media Access Control interface (MMAC) and send/receive functions for synchronization messages

NOTE Invoking of MAC is according to IEEE 802.3 and therefore not specified here.

- SYNC_SHIMP to add precise time stamps for synchronization

The Interface between the service handler (LMPM), forwarding machines and Mapping to Media Access Control (MMAC) consists of a set of queues and global data.

Figure 41 illustrates the structuring of the protocol machines for the DMPM for a device.

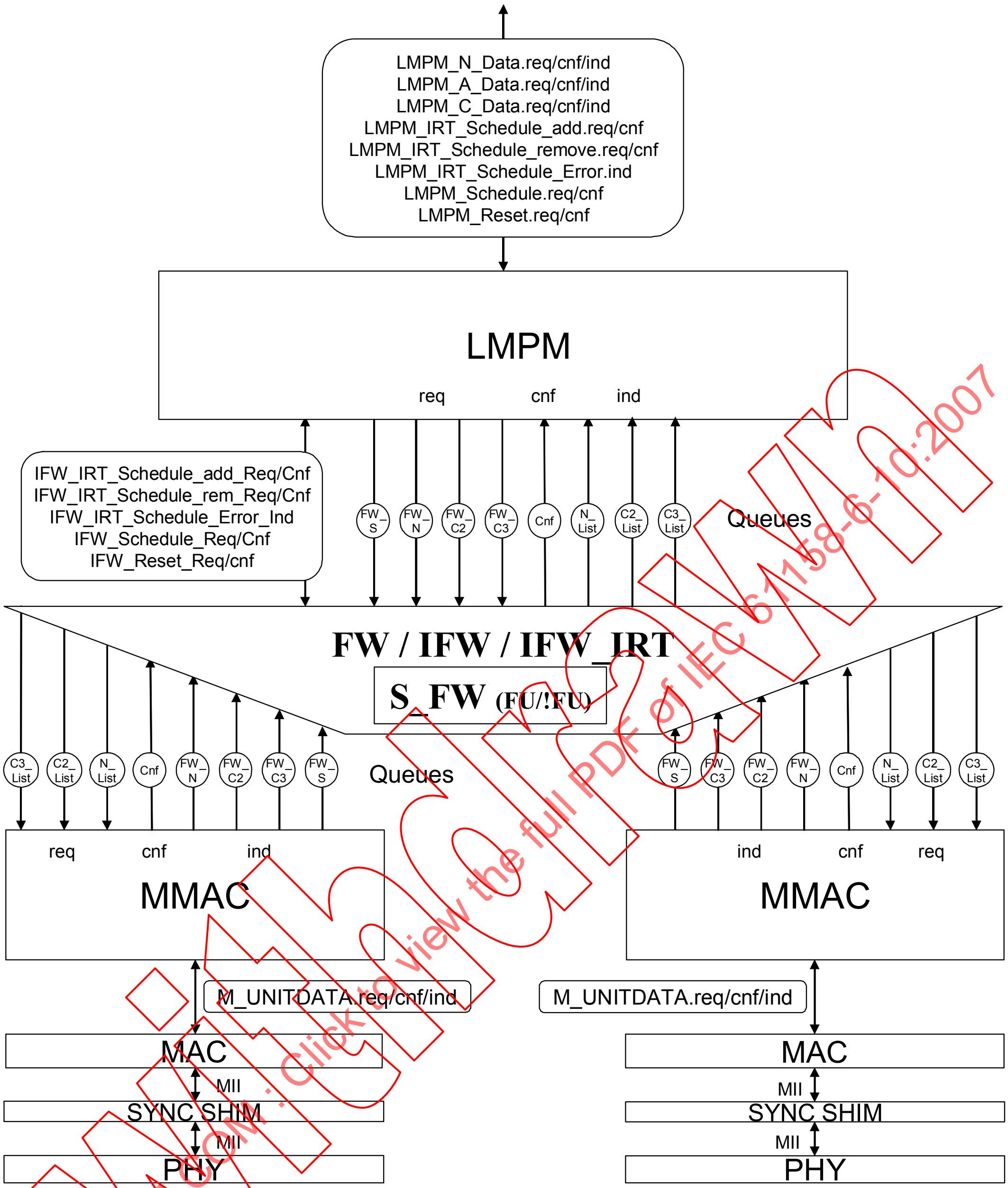

Figure 41 – Structuring of the protocol machines within the DMPM (bridge)

4.16.2 LMPM

4.16.2.1 Primitive definitions

The service primitives including their associated parameters issued by LMPM user received by LMPM and vice versa are described in the LMPM ASE in the service definition.

4.16.2.2 State machine description

The LMPM forms the interface between MMAC and LMPM-User for various data and management services. The services are all handled in the READY-State.

The service request will be validated first. A negative validation will result in a negative confirmation. Otherwise the service will be put into the appropriate priority service queue. Services executed by MMAC will be moved from the request queues to the confirmation queues. All valid incoming services will be put into the indication queue.

Local Variables

The LMPM does not use local variables. All variables are contained in the Data Resource.

4.16.2.3 LMPM state table

The LMPM State Table is shown in Table 198.

Table 198 – LMPM state table

#	Current State	Event /Condition =>Action	Next State
1	READY	**LMPM_N_Data.req (CREP, D_Port, TStamp, DA, SA, VLANPrio, VLANID, N_SDU)** /!CHECK_N_PAR => LMPM_status := IV LMPM_N_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
2	READY	**LMPM_N_Data.req (CREP, D_Port, TStamp, DA, SA, VLANPrio, VLANID, N_SDU)** /CHECK_N_PAR && !RESOURCE_N (N_SDU.Len) => LMPM_status := LS LMPM_N_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
3	READY	**LMPM_N_Data.req (CREP, D_Port, TStamp, DA, SA, VLANPrio, VLANID, N_SDU)** /CHECK_N_PAR && RESOURCE_N (N_SDU.Len) => PUT_N_FW_N_REQ (Port)	READY
4	READY	**LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VLANPrio, VLANID, A_SDU)** /!CHECK_A_PAR || !CHECK_S_PAR => LMPM_status := IV LMPM_A_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
5	READY	**LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, Prio,VLAN_Tag, LT, A_SDU)** /CHECK_A_PAR && !RESOURCE_N(A_SDU.Len) => LMPM_status := LS LMPM_A_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
6	READY	**LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VLANPrio, VLANID, A_SDU)** /CHECK_A_PAR && RESOURCE_N(A_SDU.Len) => PUT_A_FW_N_REQ (Port)	READY
7	READY	**LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, Prio,VLAN_Tag, LT, A_SDU)** /CHECK_S_PAR && !RESOURCE_S(A_SDU.Len) => LMPM_status := LS LMPM_A_Data.cnf (CREP, D_Port, TStamp, LMPM_status)	READY
8	READY	**LMPM_A_Data.req (CREP, D_Port, TStamp, DA, SA, VLANPrio, VLANID, A_SDU)** /CHECK_S_PAR && RESOURCE_S(A_SDU.Len) => PUT_A_FW_S_REQ (Port)	READY

<table>
<tr><th>#</th><th>Current State</th><th>Event
/Condition
=>Action</th><th>Next State</th></tr>
<tr><td>9</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/!CHECK_C1_PAR
|| !CHECK_C2_PAR
|| !CHECK_C3_PAR
=>
LMPM_status := IV
LMPM_C_Data.cnf (CREP, LMPM_status)</td><td>READY</td></tr>
<tr><td>10</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/CHECK_C1_PAR
&& !RESOURCE_N(C_SDU.Len)
=>
LMPM_status := LS
LMPM_C_Data.cnf (CREP, LMPM_status)</td><td>READY</td></tr>
<tr><td>11</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/CHECK_C1_PAR
&& RESOURCE_N(C_SDU.Len)
=>
PUT_C1_FW_N_REQ (Port)</td><td>READY</td></tr>
<tr><td>12</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/CHECK_C2_PAR
&& !RESOURCE_C2 (C_SDU.Len)
=>
LMPM_status := LS
LMPM_C_Data.cnf (CREP, LMPM_status)</td><td>READY</td></tr>
<tr><td>13</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/CHECK_C2_PAR
&& RESOURCE_C2 (C_SDU.Len)
=>
PUT_FW_C2_REQ (Port)</td><td>READY</td></tr>
<tr><td>14</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/CHECK_C3_PAR
&& !RESOURCE_C3 (C_SDU.Len)
=>
LMPM_status := LS
LMPM_C_Data.cnf (CREP, LMPM_status)</td><td>READY</td></tr>
<tr><td>15</td><td>READY</td><td>LMPM_C_Data.req (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)
/CHECK_C3_PAR
&& RESOURCE_C3 (C_SDU.Len)
=>
PUT_FW_C3_REQ (Port)</td><td>READY</td></tr>
<tr><td>16</td><td>READY</td><td>/CNF_List.Num_entry <> 0
&& CNF_List.First_entry.Function == N
&& CNF_List.First_entry != RT
=>
GET_CNF ()
LMPM_N_Data.cnf (CREP, D_Port, TStamp, LMPM_status)</td><td>READY</td></tr>
<tr><td>17</td><td>READY</td><td>/CNF_List.Num_entry <> 0
&& CNF_List.First_entry.Function == N
&& CNF_List.First_entry == RT
&& CNF_List.First_entry.DLSDU.FrameID >= 0x80
&& CNF_List.First_entry.DLSDU.FrameID < 0xC000
=>
GET_CNF ()
LMPM_status := WRONG_ID
LMPM_N_Data.cnf (CREP, D_Port, TStamp, LMPM_status)</td><td>READY</td></tr>
</table>

#	Current State	Event /Condition =>Action	Next State
18	READY	/CNF_List.Num_entry<>0 && CNF_List.First_entry.Function == N && CNF_List.First_entry == RT && ((CNF_List.First_entry.DLSDU.FrameID >= 0x00 && CNF_List.First_entry.DLSDU.FrameID < 0x80)) || (CNF_List.First_entry.DLSDU.FrameID >= 0xFC00 && CNF_List.First_entry.DLSDU.FrameID <= 0xFFFF)) => GET_CNF() LMPM_A_Data.cnf (CREP, S_Port, TStamp, LMPM_status)	READY
19	READY	/CNF_List.Num_entry<>0 && CNF_List.First_entry.Function == N && CNF_List.First_entry == RT && CNF_List.First_entry.DLSDU.FrameID >= 0xC000 && CNF_List.First_entry.DLSDU.FrameID < 0xFC00 => GET_CNF() LMPM_C_Data.cnf (CREP, LMPM_status)	READY
20	READY	/CNF_List.Num_entry <> 0 && CNF_List.First_entry.Function == C2 => GET_CNF () LMPM_C_Data.cnf (CREP, LMPM_status)	READY
21	READY	/CNF_List.Num_entry <> 0 && CNF_List.First_entry.Function = C3 => GET_CNF () LMPM_C_Data.cnf (CREP, LMPM_status)	READY
22	READY	/N_List.Num_entry <> 0 && N_List.First_Entry.LT != RT => GET_N_IND_N_LIST () LMPM_N_Data.ind (CREP, S_Port, TStamp, DA, SA, VLANPrio, VLANID, N_SDU)	READY
23	READY	/N_List.Num_entry <> 0 && N_List.First_Entry.LT == RT && ((N_List.First_entry.DLSDU.FrameID >= 0x00 && N_List.First_entry.DLSDU.FrameID < 0x80)) || (N_List.First_entry.DLSDU.FrameID >= 0xFC00 && N_List.First_entry.DLSDU.FrameID <= 0xFFFF)) => GET_A_IND_N_LIST () LMPM_A_Data.ind (CREP, S_Port, TStamp, DA, SA, VLANPrio, VLANID, A_SDU)	READY
24	READY	/N_List.Num_entry <> 0 && N_List.First_Entry.LT == RT && CNF_List.First_entry.DLSDU.FrameID >= 0x80 && CNF_List.First_entry.DLSDU.FrameID < 0xFC00 => GET_C_IND_N_LIST () LMPM_C_Data.ind (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)	READY
25	READY	/C2_List.Num_entry <> 0 => GET_C_IND_C2_LIST () LMPM_C_Data.ind (CREP, DA, SA, VLANPrio, VLANID, C_SDU, APDU_Status)	READY

#	Current State	Event /Condition =>Action	Next State
26	READY	/C3_List.Num_entry <> 0 => GET_C_IND_C3_LIST () LMPM_C_Data.ind (CREP, DA, SA, VLANPrio, VLANID, LT, C_SDU, APDU_status)	READY
27	READY	**LMPM_IRT_Schedule_add.req (CREP, Port, ReductionRatio, Phase)** /Phase == 0 => LMPM_status := PHASE_ERROR LMPM_Schedule_add.cnf (CREP, Port, LMPM_status)	READY
28	READY	**LMPM_IRT_Schedule_add.req (CREP, Port, ReductionRatio, Phase)** /Phase != 0 && (Phase < ReductionRatio) && (ReductionRatio > 0) && (ReductionRatio <= MAX_REDACTION_RATIO) => IFW_IRT_Schedule_add_Req (CREP, Port, ReductionRatio, Phase)	READY
29	READY	**LMPM_IRT_Schedule_add.req (CREP, Port, ReductionRatio, Phase)** /Phase != 0 && ((Phase >= ReductionRatio) || (ReductionRatio == 0) || (ReductionRatio > MAX_REDACTION_RATIO)) => LMPM_status := RATIO_ERROR LMPM_Schedule_add.cnf (CREP, Port, LMPM_status)	READY
30	READY	**IFW_IRT_Schedule_add_Cnf (CREP, Port, Status)** => LMPM_status := Status LMPM_Schedule_add.cnf (CREP, Port, LMPM_status)	READY
31	READY	**LMPM_IRT_Schedule_remove.req (CREP, Port, ReductionRatio, Phase)** => IFW_IRT_Schedule_rem_Req (CREP, Port, ReductionRatio, Phase)	READY
32	READY	**IFW_IRT_Schedule_rem_Cnf (CREP, Port, Status)** => LMPM_status := Status LMPM_Schedule_remove.cnf (CREP, Port, LMPM_status)	READY
33	READY	**IFW_IRT_Schedule_Error_Ind(CREP, ErrCode)** => LMPM_status := ErrCode LMPM_IRT_Schedule_Error.ind (CREP, LMPM_status)	READY
34	READY	**LMPM_Schedule.req (Port, ClockTime, Period, Len)** => IFW_Schedule_Req (Port, ClockTime, Period, Len)	READY
35	READY	**IFW_Schedule_Cnf (Status)** => LMPM_status := Status LMPM_Schedule.cnf (LMPM_status)	READY
36	READY	**LMPM_Reset.req ()** => RESET_FW_IND_CNF_LISTS IFW_Reset_Req ()	READY
37	READY	**IFW_Reset_Cnf (Status)** => LMPM_status := Status LMPM_Reset.cnf (LMPM_status)	READY

4.16.2.4 Macros

The LMPM macros are summarized in Table 199.

NOTE rt_id is an alias for FrameID.

Table 199 – LMPM macros table

Function name	Operations
ReducMax	Greatest suported ReductionRatio
CHECK_N_PAR	Check that all parameters of Service LMPM_N_Data.req are valid. Number of parameters must be 7. SA shall be the one of own station address N_SDU.Len: 0 … 1 500 N_SDU.Data: according to DLSDU.Len
RESOURCE_N (Len)	Check that local resources are available to handle the requested data
PUT_N_FW_N_REQ (Port)	Put a service request to the high prior service queue to FW. with FW_N_List of local Port FW_N_List.Insert() FW_N_List.Last_Entry.Function := N FW_N_List.Last_Entry.CREP := CREP FW_N_List.Last_Entry.S_Port := LOCAL FW_N_List.Last_Entry.D_Port := D_Port FW_N_List.Last_Entry. TStamp := TStamp FW_N_List.Last_Entry.DA := DA FW_N_List.Last_Entry.SA := SA FW_N_List.Last_Entry.VLANPrio := VLANPrio FW_N_List.Last_Entry.VLANID := VLANID FW_N_List.Last_Entry.LT := N_SDU.LT FW_N_List.Last_Entry.DLSDU := N_SDU FW_N_List.Num_entry++
CHECK_A_PAR	Check that all parameters of Service LMPM_A_Data.req are valid. Number of parameters must be 4. SA shall be the one of own station address A_SDU.LT == RT (A_SDU.FrameID >= 0x40 && A_SDU.FrameID < 0x80) || (A_SDU.FrameID >= 0xFC00 && A_SDU.FrameID < 0xFF00) || (A_SDU.FrameID >= 0xFC40 && A_SDU.FrameID < 0xFF00) || (A_SDU.FrameID >= 0xFC40 && A_SDU.FrameID <= 0xFFFF) A_SDU.Len: 0 … 1 440 A_SDU.Data: according to DLSDU.Len
PUT_A_FW_N_REQ (Port)	Put a service request to the high prior service queue to FW. with FW_N_List of local Port FW_N_List.Insert() FW_N_List.Last_Entry.Function := N FW_N_List.Last_Entry.CREP := CREP FW_N_List.Last_Entry.S_Port := LOCAL FW_N_List.Last_Entry.D_Port := D_Port FW_N_List.Last_Entry. TStamp := TStamp FW_N_List.Last_Entry.DA := DA FW_N_List.Last_Entry.SA := SA FW_N_List.Last_Entry.VLANPrio := VLANPrio FW_N_List.Last_Entry.VLANID := VLANID FW_N_List.Last_Entry.LT := A_SDU.LT FW_N_List.Last_Entry.DLSDU := A_SDU FW_N_List.Num_entry++

Function name	Operations
CHECK_S_PAR	Check that all parameters of Service LMPM_A_Data.req are valid. Number of parameters must be 4. SA shall be the one of own station address A_SDU.LT == RT (A_SDU.FrameID >= 0x00 && A_SDU.FrameID < 0x40) || (A_SDU.FrameID >= 0xFF00 && A_SDU.FrameID < 0xFF20) A_SDU.Len: 0 … 1 440 A_SDU.Data: according to DLSDU.Len
RESOURCE_S (Len)	Check that local resources are available to handle the requested data
PUT_A_FW_S_REQ (Port)	Put a service request to the high prior service queue to FW. with FW_S_List of local Port FW_S_List.Insert() FW_S_List.Last_Entry.Function := S FW_S_List.Last_Entry.CREP := CREP FW_S_List.Last_Entry.S_Port := LOCAL FW_S_List.Last_Entry.D_Port := D_Port FW_S_List.Last_Entry. TStamp := TStamp FW_S_List.Last_Entry.LD_Status := OK FW_S_List.Last_Entry.DA := DA FW_S_List.Last_Entry.SA := SA FW_S_List.Last_Entry.VLANPrio := VLANPrio FW_S_List.Last_Entry.VLANID := VLANID FW_S_List.Last_Entry.LT := A_SDU.LT FW_S_List.Last_Entry.DLSDU := A_SDU FW_S_List.Num_entry++
CHECK_C1_PAR	Check that all parameters of Service LMPM_C_Data.req are valid. Number of parameters must be 7. SA shall be the one of own station address C_SDU.LT == RT C_SDU.FrameID >= 0xC000 && C_SDU.FrameID < 0xFC00 C_SDU.Len: 0 .. 1 440 C_SDU.Data: according to DLSDU.Len
PUT_C1_FW_N_REQ (Port)	Put a service request to the high prior service queue to FW. with FW_N_List of local Port FW_N_List.Insert() FW_N_List.Last_Entry.Function := N FW_N_List.Last_Entry.CREP := CREP FW_N_List.Last_Entry.S_Port := LOCAL FW_N_List.Last_Entry.D_Port := AUTO FW_N_List.Last_Entry. TStamp := NIL FW_N_List.Last_Entry.DA := DA FW_N_List.Last_Entry.SA := SA FW_N_List.Last_Entry.VLANPrio :=VLANPrio FW_N_List.Last_Entry.VLANID := VLANID FW_N_List.Last_Entry.LT := C_SDU.LT FW_N_List.Last_Entry.DLSDU := C_SDU, APDU_Status FW_N_List.Num_entry++
CHECK_C2_PAR	Check that all parameters of Service LMPM_C_Data.req are valid. Number of parameters must be 7. SA shall be the one of own station address C_SDU.LT == RT C_SDU.FrameID >= 0x8000 && C_SDU.FrameID < 0xC000 C_SDU.Len: 0 .. 1 440 C_SDU.Data: according to DLSDU.Len
RESOURCE_C2 (Len)	Check that local resources are available to handle the requested data

Function name	Operations
PUT_FW_C2_REQ (Port)	Put a service request to the high prior service queue to FW. with FW_C2_List of local Port FW_C2_List.Insert() FW_C2_List.Last_Entry.Function := C2 FW_C2_List.Last_Entry.CREP := CREP FW_C2_List.Last_Entry.S_Port := LOCAL FW_C2_List.Last_Entry.D_Port := AUTO FW_C2_List.Last_Entry. TStamp := NIL FW_C2_List.Last_Entry.DA := DA FW_C2_List.Last_Entry.SA := SA FW_C2_List.Last_Entry.VLANPrio := VLANPrio FW_C2_List.Last_Entry.VLANID := VLANID FW_C2_List.Last_Entry.LT := C_SDU.LT FW_C2_List.Last_Entry.DLSDU := C_SDU, APDU_Status FW_C2_List.Num_entry++
CHECK_C3_PAR	Check that all parameters of Service LMPM_C_Data.req are valid. Number of parameters must be 7. SA shall be the one of own station address C_SDU.LT == RT C_SDU.FrameID >= 0x0100 && C_SDU.FrameID < 0x8000 C_SDU.Len: 0 ... 1 440 C_SDU.Data: according to DLSDU.Len
RESOURCE_C3 (Len)	Check that local resources are available to handle the requested data
PUT_FW_C3_REQ (Port)	Put a service request to the high prior service queue to FW. with FW_C3_List of local Port FW_C3_List.Insert() FW_C2_List.Last_Entry.Function := C3 FW_C3_List.Last_Entry.CREP := CREP FW_C3_List.Last_Entry.S_Port := LOCAL FW_C3_List.Last_Entry.D_Port := AUTO FW_C3_List.Last_Entry. TStamp := NIL FW_C3_List.Last_Entry.DA := DA FW_C3_List.Last_Entry.SA := SA FW_C3_List.Last_Entry.VLANPrio := VALNPrio FW_C3_List.Last_Entry.VLANID := NIL FW_C3_List.Last_Entry.LT := C_SDU.LT FW_C3_List.Last_Entry.DLSDU := C_SDU, APDU_Status FW_C3_List.Num_entry++
GET_CNF()	Get a service confirmation from the confirmation queue of IFW. CNF_List.Num_entry-- CREP := CNF_List.First_Entry.CREP LMPM_status := CNF_List.Last_Entry.Status D:Port := CNF_List.Last_Entry.D_Port TStamp := CNF_List.Last_Entry. TStamp CNF_List.Remove()

Function name	Operations
GET_N_IND_N_LIST()	Get a service indication from the indication queue of IFW. N_List.Num_entry-- CREP := N_List.First_Entry.CREP S_Port := N_List.First_Entry.S_Port TStamp := N_List.First_Entry. TStamp DA := N_List.First_Entry.DA SA := N_List.First_Entry.SA VALNPrio := N_List.First_Entry.VLANPrio VLANID := N_List.First_Entry.VLANID N_SDU := N_List.First_Entry.DLSDU N_List.Remove()
GET_A_IND_N_LIST()	Get a service indication from the indication queue of IFW. N_List.Num_entry-- S_Port := N_List.First_Entry.S_Port TStamp := N_List.First_Entry. TStamp DA := N_List.First_Entry.DA SA := N_List.First_Entry.SA VLANPrio := N_List.First_Entry.VLANPrio VLANID := N_List.First_Entry.VLANID A_SDU := N_List.First_Entry.DLSDU N_List.Remove()
GET_C_IND_N_LIST()	Get a service indication from the indication queue of IFW. N_List.Num_entry-- DA := N_List.First_Entry.DA SA := N_List.First_Entry.SA VALNPrio := N_List.First_Entry.VLANPrio VLANID := N_List.First_Entry.VLANID C_SDU, APDU_Status := N_List.First_Entry.DLSDU N_List.Remove()
GET_C_IND_C2_LIST()	Get a service indication from the indication queue of IFW. C2_List.Num_entry-- DA := C2_List.First_Entry.DA SA := C2_List.First_Entry.SA VLANPrio := C2_List.First_Entry.VLANPrio VLANID := C2_List.First_Entry.VLANID LT := C2_List.First_Entry.LT C_SDU, APDU_Status := C2_List.First_Entry.DLSDU C2_List.Remove()
GET_C_IND_C3_LIST()	Get a service indication from the indication queue of IFW. C3_List.Num_entry-- DA := C3_List.First_Entry.DA SA := C3_List.First_Entry.SA VLANPrio := C3_List.First_Entry.VLANPrio VLANID := C3_List.First_Entry.VLANID C_SDU, APDU_Status := C3_List.First_Entry.DLSDU C3_List.Remove()
RESET_FW_IND_CNF_LISTS	Reset all Lists with queued services to LMPM in the data interface to IFW sublayer FW_N_List.Num_entry-- FW_N_List.Remove() until empty FW_C2_List.Num_entry-- FW_C2_List.Remove() until empty FW_C3_List.Num_entry-- FW_C3_List.Remove() until empty N_List.Num_entry-- N_List.Remove() until empty C2_List.Num_entry-- C2_List.Remove() until empty C3_List.Num_entry-- C3_List.Remove() until empty CNF_List.Num_entry-- CNF_List.Remove() until empty

4.16.2.5 Functions

Table 200 contains the functions used by the LMPM and their descriptions.

Table 200 – LMPM function table

Name	Meaning
LMPM_A_Data	Handle RTA_CLASS_x requests, confirmations and indications and their queuing.
LMPM_N_Data	Handle other requests, confirmations and indications and their queuing.
LMPM_C_Data	Handle RT_CLASS_x requests, confirmations and indications and their queuing.
LMPM_IRT_Schedule_add	Handle RT_CLASS_3 requests for each port.
LMPM_IRT_Schedule_remove	Handle RT_CLASS_3 requests for each port.
LMPM_IRT_Schedule_Error	Handle RT_CLASS_3 errors for each port.
LMPM_Schedule	Handles the Transmission Period. It is called by the PTCP synchronized clock, delivering Transmission Period, and Clock Domain Phase Counter.
LMPM_Time_Event	Delivers Transmission Period, Clock Domain Phase Counter, and the source (LOCAL or REMOTE) of this information. It is called by the PTCP synchronized clock.
LMPM_Reset	Remove all entries from all queues.

5 Application layer protocol specification for distributed automation

5.1 FAL syntax description

5.1.1 IDL specification

5.1.1.1 Basic information

The IDL specification provides the formal specification of the FAL service parameters with their data types. This information is used for data marshaling by the ORPC wire protocol to build the APDU. Therefore, the IDL specification describes the format, the order and the content of the APDU. The encoding of the basic data types itself belongs to the used ORPC wire protocol and is part of the underlying abstract ORPC model.

5.1.1.2 Error messages

Table 201 shows the specific error messages (HRESULT data type) for the FAL AE. Error messages are 32 bit values laid out as shown in Figure 42.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
S	R		Facility													Code															

where

S - (1 bit) is the severity code

0 – Success

1 – Error

R - (2 bits) reserved, to be set to zero

Facility - (13 bits) is the facility code

Code - (16 bits) is the facility's status code

Figure 42 – Error message structure

A particular HRESULT value by convention uses the following naming structure:

CBA_<SEV: S or E>_<Reason>.

Table 201 – Error messages

Value	Code	Description
CBA_E_ACCESSBLOCKED	0x8004CB15L	Access to this item is blocked because a constant is currently assigned to it.
CBA_E_CAPACITYEXCEEDED	0x8004CB1DL	The request exceeds the resources according to the machines capacity.
CBA_E_COUNTEXCEEDED	0x8004CB16L	The request exceeds the resources according to their count.
CBA_E_CRDATALENGTH	0x8004CB1BL	The length of the requested data for this CR is invalid for RT.
CBA_E_DEFECT	0x8004CB11L	Hardware defect detected, replacement needed.
CBA_E_DISCONNECTRUNNING	0x8004CB21L	The request exceeds the resources and there is a Disconnect running which will free some resources later.
CBA_E_FLAGUNSUPPORTED	0x8004CB1CL	One of the flags used are not supported in this implementation.
CBA_E_FRAMECOUNTUNSUPPORTED	0x8004CB23L	The number of RT frames transferred between two ACCOs at the same QoS is restricted by the implementation.
CBA_E_INUSE	0x8004CB0EL	The destination specified in the item identifier is already connected to some other provider.
CBA_E_INVALIDCONNECTION	0x8004CB08L	It is not allowed to specify a connection from an identifier to itself.
CBA_E_INVALIDCOOKIE	0x8004CB09L	The Cookie value is not valid.
CBA_E_INVALIDENUMVALUE	0x8004CB04L	The enumeration value is invalid.
CBA_E_INVALIDEPSILON	0x8004CB06L	The Epsilon type or value is not valid.
CBA_E_INVALIDID	0x8004CB05L	The ConsumerID or ProviderID is not valid.
CBA_E_INVALIDSUBSTITUTE	0x8004CB07L	The Substitute type or value is not valid.
CBA_E_ITEMTOOLARGE	0x8004CB1AL	The size of the item exceeds the upper limit for RT.
CBA_E_LIMITVIOLATION	0x8004CB12L	The data exceeds limits of the corresponding data type.
CBA_E_LINKFAILURE	0x8004CB24L	The network cards current link status is unsuitable for this protocol.
CBA_E_LOCATIONCHANGED	0x8004CB22L	The location of the restored connection differs from the location of the saved one.
CBA_E_MALFORMED	0x8004CB00L	The identifier is malformed (too long, unallowed characters, no separation characters, syntactically invalid)
CBA_E_MODECHANGE	0x8004CB25L	The consumer tried to change the data transfer mode from push (OnData-Changed) to pull (GetConnectionData) or vice versa.
CBA_E_NONACCESSIBLE	0x8004CB10L	The item is not accessible.
CBA_E_NOTAPPLICABLE	0x8004CB0FL	The operation is currently not applicable.
CBA_E_NOTROUTABLE	0x8004CB20L	Routing is forbidden for this protocol and the remote station can't be reached via one of the local subnets.
CBA_E_OUTOFACCOPAIRS	0x8004CB19L	The request exceeds the resources according to the AccoPairs.
CBA_E_OUTOFPARTNERACCOS	0x8004CB18L	The request exceeds the resources according to the partner ACCOs.
CBA_E_PERSISTRUNNING	0x8004CB0DL	While the Save service is running, no changes are allowed to the configuration data base (try again in a few seconds).

Value	Code	Description
CBA_E_QCNOTAPPLICABLE	0x8004CB14L	The quality code is not applicable for this operation.
CBA_E_QOSTYPENOTAPPLICABLE	0x8004CB13L	The QoS type is not applicable for this operation.
CBA_E_QOSTYPEUNSUPPORTED	0x8004CB0BL	The QoS type is unsupported.
CBA_E_QOSVALUEUNSUPPORTED	0x8004CB0CL	The QoS value is unsupported.
CBA_E_SIZEEXCEEDED	0x8004CB17L	The request exceeds the resources according to their size.
CBA_E_STATIONFAILURE	0x8004CB1FL	The remote station failed regarding the data transfer.
CBA_E_SUBELEMENTMISMATCH	0x8004CB1EL	The sub element access list does not match the item type.
CBA_E_TIMEVALUEUNSUPPORTED	0x8004CB0AL	The time value is unsupported.
CBA_E_TYPEMISMATCH	0x8004CB03L	The type specified does not match the expected type.
CBA_E_UNKNOWNMEMBER	0x8004CB02L	The member specified in the item identifier is not known.
CBA_E_UNKNOWNOBJECT	0x8004CB01L	The object specified in the item identifier is not known.
CBA_S_ESTABLISHING	0x0004CA01L	The connection is not yet established.
CBA_S_FRAMEEMPTY	0x0004CA06L	RT Frame is empty, e.g. carries no connections.
CBA_S_NOCONNECTION	0x0004CA02L	There is no connection from the provider to this specific consumer.
CBA_S_NOCONNECTIONDATA	0x0004CA05L	Connection data is currently not received.
CBA_S_PERSISTPENDING	0x0004CA00L	The value requested is currently not persistent.
CBA_S_VALUEBUFFERED	0x0004CA03L	The value was only buffered and has no immediate effect on the process (e.g. device is in state CBAReady).
CBA_S_VALUEUNCERTAIN	0x0004CA04L	It is uncertain, whether the value is valid, since the device is currently in state CBAReady.
DISP_E_BADINDEX	0x8002000BL	Invalid index.
DISP_E_BADPARAMCOUNT	0x8002000EL	The number of elements provided to DISPPARAMS is different from the number of arguments accepted by the method or property.
DISP_E_BADVARTYPE	0x80020008L	One of the arguments is not a valid variant type.
DISP_E_EXCEPTION	0x80020009L	The application needs to raise an exception. In this case, the structure passed in pExcepInfo should be filled in.
DISP_E_MEMBERNOTFOUND	0x80020003L	The requested member does not exist, or the call to Invoke tried to set the value of a read-only property.
DISP_E_NONAMEDARGS	0x80020007L	This implementation of IDispatch does not support named arguments
DISP_E_OVERFLOW	0x8002000AL	One of the arguments could not be coerced to the specified type.
DISP_E_PARAMNOTFOUND	0x80020004L	One of the parameter DISPIDs does not correspond to a parameter on the method. In this case, puArgErr should be set to the first argument that contains the error.
DISP_E_PARAMNOTOPTIONAL	0x8002000FL	A required parameter was omitted.

Value	Code	Description
DISP_E_TYPEMISMATCH	0x80020005L	One or more of the arguments could not be coerced. The index within rgvarg of the first parameter with the incorrect type is returned in the puArgErr parameter.
DISP_E_UNKNOWNINTERFACE	0x80020001L	The interface identifier passed in riid is not IID_NULL.
DISP_E_UNKNOWNLCID	0x8002000CL	Unknown language.
DISP_E_UNKNOWNNAME	0x80020006L	Unknown name.
E_ABORT	0x80004004L	
E_ACCESSDENIED	0x80070005L	
E_FAIL	0x80004005L	Unspecified error.
E_HANDLE	0x80070006L	
E_INVALIDARG	0x80070057L	One ore more arguments are invalid.
E_NOINTERFACE	0x80004002L	No such interface supported.
E_NOTIMPL	0x80004001L	The service is not implemented.
E_OUTOFMEMORY	0x8007000EL	Insufficient memory to complete the call.
E_POINTER	0x80004003L	
E_UNEXPECTED	0x8000FFFFL	
RPC_E_INVALID_OBJECT	0x80010114L	The requested object does not exist.
RPC_E_INVALID_OID	0x80070777L	The specified object was not found or recognized.
RPC_E_INVALID_OXID	0x80070776L	The object exporter was not found.
RPC_E_INVALID_SET	0x80070778L	The object exporter set was not found.
RPC_E_VERSION_MISMATCH	0x80010110L	The ORPC version on the client and server machine does not match.
RPC_S_PROCNUM_OUT_OF_RANGE	0x000006D1L	The procedure number is out of range.
S_FALSE	0x00000001L	Success but with additional results, "function worked and the result is false".
S_OK	0x00000000L	Success, "everything worked".
TYPE_E_ELEMENTNOTFOUND	0x8002802BL	Element not found.

5.1.1.3 Interface definitions for the FAL AE

5.1.1.3.1 Overview

The IDL description for the FAL AE, the abstract classes and the used type definitions are defined below. The types are composed of enumerations and structures of other types. These types are used by the FAL services defined in the interface description to store data into a classified and known container or to use quality codes in a readable form. In the class definitions are the interfaces summarized.

5.1.1.3.2 Coding of FAL ASE data types

5.1.1.3.2.1 VARTYPE

The allowed values of VARTYPE are shown in Table 202.

Table 202 – VARTYPE values

Value	Code
VT_EMPTY	0x0000
VT_NULL	0x0001
VT_I2	0x0002
VT_I4	0x0003
VT_R4	0x0004
VT_R8	0x0005
VT_DATE	0x0007
VT_BSTR	0x0008
VT_DISPATCH	0x0009
VT_ERROR	0x000A
VT_BOOL	0x000B
VT_UNKNOWN	0x000D
VT_I1	0x0010
VT_UI1	0x0011
VT_UI2	0x0012
VT_UI4	0x0013
VT_USERDEFINED	0x001D
VT_SAFEARRAY_I2	0x2002
VT_SAFEARRAY_I4	0x2003
VT_SAFEARRAY_R4	0x2004
VT_SAFEARRAY_R8	0x2005
VT_SAFEARRAY_DATE	0x2007
VT_SAFEARRAY_BSTR	0x2008
VT_SAFEARRAY_ERROR	0x200A
VT_SAFEARRAY_BOOL	0x200B
VT_SAFEARRAY_I1	0x2010
VT_SAFEARRAY_UI1	0x2011
VT_SAFEARRAY_UI2	0x2012
VT_SAFEARRAY_UI4	0x2013

5.1.1.3.2.2 ITEMQUALITYDEF

This data type contains the status information of the related data. It consists of three portions: Quality, Substatus and Limits. There are four states of quality (Bad, Uncertain, Good (Non Cascade) and Good (Cascade)), a set of sub-status values for each quality, and four states of the limits (OK, low limited (LL), high limited (HL) and constant (C)). The allowed values are shown in Table 203. The detailed encoding of the quality code is shown in Figure 43.

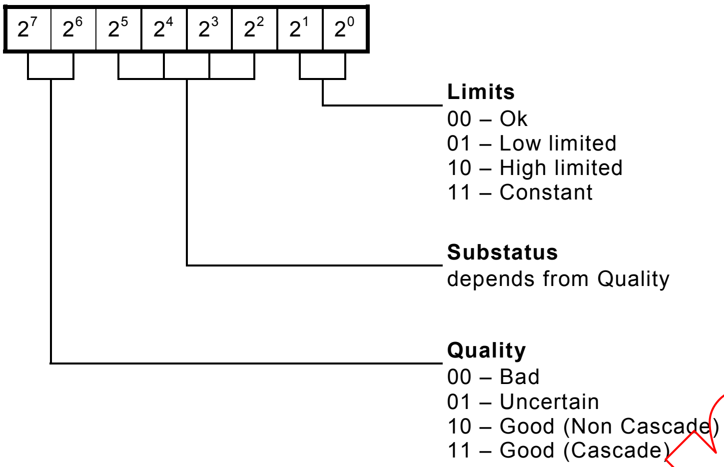

Figure 43 – Coding scheme of ITEMQUALITYDEF

Table 203 – ITEMQUALITYDEF values

Value	Code
BadNonSpecific	0x00
BadNonSpecificLL	0x01
BadNonSpecificHL	0x02
BadNonSpecificC	0x03
BadConfigurationError	0x04
BadConfigurationErrorLL	0x05
BadConfigurationErrorHL	0x06
BadConfigurationErrorC	0x07
BadNotConnected	0x08
BadNotConnectedLL	0x09
BadNotConnectedHL	0x0A
BadNotConnectedC	0x0B
BadDeviceFailure	0x0C
BadDeviceFailureLL	0x0D
BadDeviceFailureHL	0x0E
BadDeviceFailureC	0x0F
BadSensorFailure	0x10
BadSensorFailureLL	0x11
BadSensorFailureHL	0x12
BadSensorFailureC	0x13
BadLastKnownValue	0x14
BadLastKnownValueLL	0x15
BadLastKnownValueHL	0x16
BadLastKnownValueC	0x17
BadCommFailure	0x18
BadCommFailureLL	0x19
BadCommFailureHL	0x1A
BadCommFailureC	0x1B
BadOutOfService	0x1C
BadOutOfServiceLL	0x1D

Value	Code
BadOutOfServiceHL	0x1E
BadOutOfServiceC	0x1F
UncertainNonSpecific	0x40
UncertainNonSpecificLL	0x41
UncertainNonSpecificHL	0x42
UncertainNonSpecificC	0x43
UncertainLastUsableValue	0x44
UncertainLastUsableValueLL	0x45
UncertainLastUsableValueHL	0x46
UncertainLastUsableValueC	0x47
UncertainSubstituteSet	0x48
UncertainSubstituteSetLL	0x49
UncertainSubstituteSetHL	0x4A
UncertainSubstituteSetC	0x4B
UncertainInitialValue	0x4C
UncertainInitialValueLL	0x4D
UncertainInitialValueHL	0x4E
UncertainInitialValueC	0x4F
UncertainSensorNotAccurate	0x50
UncertainSensorNotAccurateLL	0x51
UncertainSensorNotAccurateHL	0x52
UncertainSensorNotAccurateC	0x53
UncertainEngineeringUnitsExceeded	0x54
UncertainEngineeringUnitsExceededLL	0x55
UncertainEngineeringUnitsExceededHL	0x56
UncertainEngineeringUnitsExceededC	0x57
UncertainSubNormal	0x58
UncertainSubNormalLL	0x59
UncertainSubNormalHL	0x5A
UncertainSubNormalC	0x5B
UncertainConfigurationError	0x5C
UncertainConfigurationErrorLL	0x5D
UncertainConfigurationErrorHL	0x5E
UncertainConfigurationErrorC	0x5F
UncertainSimulatedValue	0x60
UncertainSimulatedValueLL	0x61
UncertainSimulatedValueHL	0x62
UncertainSimulatedValueC	0x63
UncertainSensorCalibration	0x64
UncertainSensorCalibrationLL	0x65
UncertainSensorCalibrationHL	0x66
UncertainSensorCalibrationC	0x67
GoodNonCascOk	0x80
GoodNonCascOkC	0x83
GoodNonCascActiveUpdateEvent	0x84

Value	Code
GoodNonCascActiveUpdateEventLL	0x85
GoodNonCascActiveUpdateEventHL	0x86
GoodNonCascActiveUpdateEventC	0x87
GoodNonCascActiveAdvisoryAlarm	0x88
GoodNonCascActiveAdvisoryAlarmLL	0x89
GoodNonCascActiveAdvisoryAlarmHL	0x8A
GoodNonCascActiveAdvisoryAlarmC	0x8B
GoodNonCascActiveCriticalAlarm	0x8C
GoodNonCascActiveCriticalAlarmLL	0x8D
GoodNonCascActiveCriticalAlarmHL	0x8E
GoodNonCascActiveCriticalAlarmC	0x8F
GoodNonCascUnackUpdateEvent	0x90
GoodNonCascUnackUpdateEventLL	0x91
GoodNonCascUnackUpdateEventHL	0x92
GoodNonCascUnackUpdateEventC	0x93
GoodNonCascUnackAdvisoryAlarm	0x94
GoodNonCascUnackAdvisoryAlarmLL	0x95
GoodNonCascUnackAdvisoryAlarmHL	0x96
GoodNonCascUnackAdvisoryAlarmC	0x97
GoodNonCascUnackCriticalAlarm	0x98
GoodNonCascUnackCriticalAlarmLL	0x99
GoodNonCascUnackCriticalAlarmHL	0x9A
GoodNonCascUnackCriticalAlarmC	0x9B
GoodNonCascInitialFailSafe	0xA0
GoodNonCascInitialFailSafeLL	0xA1
GoodNonCascInitialFailSafeHL	0xA2
GoodNonCascInitialFailSafeC	0xA3
GoodNonCascMaintenanceRequired	0xA4
GoodNonCascMaintenanceRequiredLL	0xA5
GoodNonCascMaintenanceRequiredHL	0xA6
GoodNonCascMaintenanceRequiredC	0xA7
GoodCascOk	0xC0
GoodCascOkC	0xC3
GoodCascInitializationAcknowledge	0xC4
GoodCascInitializationAcknowledgeLL	0xC5
GoodCascInitializationAcknowledgeHL	0xC6
GoodCascInitializationAcknowledgeC	0xC7
GoodCascInitializationRequest	0xC8
GoodCascInitializationRequestLL	0xC9
GoodCascInitializationRequestHL	0xCA
GoodCascInitializationRequestC	0xCB
GoodCascNotInvited	0xCC
GoodCascNotInvitedLL	0xCD
GoodCascNotInvitedHL	0xCE
GoodCascNotInvitedC	0xCF

Value	Code
GoodCascDoNotSelect	0xD4
GoodCascDoNotSelectLL	0xD5
GoodCascDoNotSelectHL	0xD6
GoodCascDoNotSelectC	0xD7
GoodCascLocalOverride	0xD8
GoodCascLocalOverrideLL	0xD9
GoodCascLocalOverrideHL	0xDA
GoodCascLocalOverrideC	0xDB
GoodCascInitiateFailSafe	0xE0
GoodCascInitiateFailSafeLL	0xE1
GoodCascInitiateFailSafeHL	0xE2
GoodCascInitiateFailSafeC	0xE3

5.1.1.3.2.3 STATEDEF

The allowed values of STATEDEF are shown in Table 204.

Table 204 – STATEDEF values

Value	Code
CBANonExistent	0x00
CBAInitializing	0x01
CBAReady	0x02
CBAOperating	0x03
CBADefect	0x04

5.1.1.3.2.4 GROUPERRORDEF

The allowed values of GROUPERRORDEF are shown in Table 205.

Table 205 – GROUPERRORDEF values

Value	Code
CBANonAccessible	0x00
CBAOkay	0x01
CBAProblem	0x02
CBAUnknown	0x03

5.1.1.3.2.5 ACCESSRIGHTSDEF

The allowed values of ACCESSRIGHTSDEF are shown in Table 206.

Table 206 – ACCESSRIGHTSDEF values

Value	Code
CBANoAccess	0x00
CBAReadAccess	0x01
CBAWriteAccess	0x02
CBAFullAccess	0x03

5.1.1.3.2.6 PERSISTDEF

The allowed values of PERSISTDEF are shown in Table 207.

Table 207 – PERSISTDEF values

Value	Code
CBAVolatile	0x00
CBAPendingPersistent	0x01
CBAPersistent	0x02

5.1.1.3.2.7 READITEMOUT

```
typedef struct READITEMOUT
{
              VARIANT          Value;               // value
              ITEMQUALITYDEF   QualityCode;         // quality code
              FILETIME         TimeStamp;           // time stamp
              HRESULT          ErrorState;          // partial result
} READITEMOUT, *pREADITEMOUT;
```

5.1.1.3.2.8 WRITEITEMIN

```
typedef struct WRITEITEMIN
{
  [string]    LPWSTR           Item;                // item name ("Object.Member")
              VARIANT          Value;               // value
} WRITEITEMIN, *pWRITEITEMIN;
```

5.1.1.3.2.9 WRITEITEMQCDIN

```
typedef struct WRITEITEMQCDIN
{
              WRITEITEMIN      WriteItem;           // WRITEITEMIN structure
              ITEMQUALITYDEF   QualityCode;         // quality code
              FILETIME         TimeStamp;           // time stamp
} WRITEITEMQCDIN, *pWRITEITEMQCDIN;
```

5.1.1.3.2.10 ADDCONNECTIONIN

```
typedef struct ADDCONNECTIONIN
{
  [string]    LPWSTR      ProviderItem; // source item name ("Object.Member")
  [string]    LPWSTR      ConsumerItem; // destination item name ("Object.Member")
              PERSISTDEF  Persistence;      // required persistence
              VARIANT     SubstituteValue;  // substitute value
              VARIANT     Epsilon;          // epsilon value (hysteresis)
} ADDCONNECTIONIN, *pADDCONNECTIONIN;
```

5.1.1.3.2.11 ADDCONNECTIONOUT

```
typedef struct ADDCONNECTIONOUT
{
              DWORD            ConsumerID;       // consumer ID
              WORD             Version;          // version number of the connection
              HRESULT          ErrorState;       // partial result
} ADDCONNECTIONOUT, *pADDCONNECTIONOUT;
```

5.1.1.3.2.12 GETIDOUT

```
typedef struct GETIDOUT
{
```

```
                DWORD           ConsumerID;       // consumer ID
                BOOLEAN         State;            // activation state (active = TRUE)
                WORD            Version;          // version number
                HRESULT         ErrorState;       // error state of the connection
// A separate HRESULT containing the partial result is not specified,
// because partial errors are not possible here.
} GETIDOUT, *pGETIDOUT;
```

5.1.1.3.2.13 GETCONNECTIONOUT

```
typedef struct GETCONNECTIONOUT
{
  [string]     LPWSTR      Provider;      // provider name ("PDev!LDev")
  [string]     LPWSTR      ProviderItem; // source item name ("Object.Member")
  [string]     LPWSTR      ConsumerItem; // destination item name ("Object.Member")
               VARIANT     SubstituteValue;    // substitute value
               VARIANT     Epsilon;            // epsilon value (hysteresis)
               WORD        QoSType;            // quality of service type
               WORD        QoSValue;           // quality of service value
               BOOLEAN     State;              // activation state (active = TRUE)
               PERSISTDEF  Persistence;        // persistence
               WORD        Version;            // version number
               HRESULT     ErrorState;         // error state of the connection
// A separate HRESULT containing the partial result is not specified.
// That's a bug :-(
// We use this work-around: All partial errors are mapped to E_FAIL. Only
// partial errors cause the result of the GetIDOut method to become S_FALSE.
// The error states of the connections have no influence on the result of
// the method.
} GETCONNECTIONOUT, *pGETCONNECTIONOUT;
```

5.1.1.3.2.14 CONNECTIN

```
typedef struct CONNECTIN
{
  [string]     LPWSTR          ProviderItem; // source item name ("Object.Member")
               VARTYPE         DataType;     // basic data type
               VARIANT         Epsilon;      // epsilon value (hysteresis)
               DWORD           ConsumerID;   // consumer ID
} CONNECTIN, *pCONNECTIN;
```

5.1.1.3.2.15 CONNECTOUT

```
typedef struct CONNECTOUT
{
               DWORD           ProviderID;     // provider ID
               HRESULT         ErrorState;     // partial result
} CONNECTOUT, *pCONNECTOUT;
```

5.1.1.3.2.16 GETCONSCONNOUT

```
typedef struct GETCONSCONNOUT
{
  [string]     LPWSTR      Provider;      // provider name ("PDev!LDev")
  [string]     LPWSTR      ProviderItem; // source item name ("Object.Member")
  [string]     LPWSTR      ConsumerItem; // destination item name ("Object.Member")
               VARIANT     SubstituteValue;    // substitute value
               VARIANT     Epsilon;            // epsilon value (hysteresis)
               WORD        QoSType;            // quality of service type
               WORD        QoSValue;           // quality of service value
               BOOLEAN     State;              // activation state (active = TRUE)
               PERSISTDEF  Persistence;        // persistence
               HRESULT     PartialResult;      // partial result
} GETCONSCONNOUT, *pGETCONSCONNOUT;
```

5.1.1.3.2.17 DIAGCONSCONNOUT

```
typedef struct DIAGCONSCONNOUT
{
// rule of thumb: iMap uses GetConsConnections on each AccoStamp change
// and DiagConsConnections on each GroupError event, so GetConsConnections
// should include all values which effect an AccoStamp change and DiagCons-
// Connections should include all values which effect a GroupError change.
               BOOLEAN         State;          // activation state (active = TRUE)
               PERSISTDEF      Persistence;    // persistence
               WORD            Version;        // version number
               HRESULT         ErrorState;     // error state of the connection
               HRESULT         PartialResult;  // partial result
```

```
} DIAGCONSCONNOUT, *pDIAGCONSCONNOUT;
```

5.1.1.3.2.18 GETPROVCONNOUT

```
typedef struct GETPROVCONNOUT
{
  [string]    LPWSTR     Consumer;      // consumer name ("PDev!LDev")
  [string]    LPWSTR     ProviderItem;  // source item name ("Object.Member")
              DWORD      ConsumerID;    // consumer ID
              VARIANT    Epsilon;       // epsilon value (hysteresis) - DCOM only
              WORD       QoSType;       // quality of service type
              WORD       QoSValue;      // quality of service value
              BOOLEAN    State;         // activation state (active = TRUE)
              HRESULT    PartialResult; // partial result
} GETPROVCONNOUT, *pGETPROVCONNOUT;
```

5.1.1.3.2.19 CONNECTIN2

```
typedef struct CONNECTIN2
{
  [string]    LPWSTR           ProviderItem; // source item name ("Object.Member")
              WORD             TypeDescLen;  // length of type description
  [size_is(TypeDescLen)] WORD  *pTypeDesc;   // type description
              VARIANT          Epsilon;      // epsilon value (hysteresis)
              DWORD            ConsumerID;   // consumer ID
} CONNECTIN2, *pCONNECTIN2;
```

5.1.1.3.2.20 MACAddr

```
typedef struct MACAddr
{
              // 6 byte Ethernet MAC address (highest byte first)
              BYTE             B0;
              BYTE             B1;
              BYTE             B2;
              BYTE             B3;
              BYTE             B4;
              BYTE             B5;
} MACAddr, *pMACAddr;
```

5.1.1.3.2.21 CONNECTINCR

```
typedef struct CONNECTINCR
{
              WORD             ConsumerCRID;         // frame ID of the CR
              WORD             ConsumerCRLength;     // maximum length of the CR
} CONNECTINCR, *pCONNECTINCR;
```

5.1.1.3.2.22 CONNECTOUTCR

```
typedef struct CONNECTOUTCR
{
              DWORD            ProviderCRID;         // provider CR ID
              HRESULT          PartialResult;        // partial result
} CONNECTOUTCR, *pCONNECTOUTCR;
```

5.1.1.3.2.23 CONNECTINSRT

```
typedef struct CONNECTINSRT
{
  [string]    LPWSTR           ProviderItem;   // source item name ("Object.Member")
              WORD             TypeDescLen;    // length of type description
  [size_is(TypeDescLen)] WORD  *pTypeDesc;     // type description
              DWORD            ConsumerID;     // consumer ID
              WORD             Length;         // marshalled length of record
(redundant information for robustness)
} CONNECTINSRT, *pCONNECTINSRT;
```

5.1.1.3.2.24 UUID

Predefined values are shown in Table 208.

Table 208 – UUID values

Value	Code
UUID_NULL	00000000-0000-0000-0000-000000000000
UUID_IUnknown	00000000-0000-0000-C000-000000000046
UUID_IDispatch	00020400-0000-0000-C000-000000000046
UUID_ICBAPhysicalDevice	CBA00001-6C97-11D1-8271-00A02442DF7D
UUID_ICBAPhysicalDevice2	CBA00006-6C97-11D1-8271-00A02442DF7D
UUID_ICBABrowse	CBA00002-6C97-11D1-8271-00A02442DF7D
UUID_ICBABrowse2	CBA00007-6C97-11D1-8271-00A02442DF7D
UUID_ICBAPersist	CBA00005-6C97-11D1-8271-00A02442DF7D
UUID_ICBAPersist2	CBA00008-6C97-11D1-8271-00A02442DF7D
UUID_ICBALogicalDevice	CBA00011-6C97-11D1-8271-00A02442DF7D
UUID_ICBALogicalDevice2	CBA00017-6C97-11D1-8271-00A02442DF7D
UUID_ICBAState	CBA00012-6C97-11D1-8271-00A02442DF7D
UUID_ICBATime	CBA00014-6C97-11D1-8271-00A02442DF7D
UUID_ICBAGroupError	CBA00015-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoMgt	CBA00041-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoMgt2	CBA00046-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoServer	CBA00043-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoServer2	CBA00048-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoServerSRT	CBA00045-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoCallback	CBA00042-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoCallback2	CBA00047-6C97-11D1-8271-00A02442DF7D
UUID_ICBAAccoSync	CBA00044-6C97-11D1-8271-00A02442DF7D
UUID_ICBARTAuto	CBA00051-6C97-11D1-8271-00A02442DF7D
UUID_ICBARTAuto2	CBA00052-6C97-11D1-8271-00A02442DF7D
UUID_ICBASystemProperties	CBA00062-6C97-11D1-8271-00A02442DF7D
UUID_PhysicalDevice	CBA00000-6C97-11D1-8271-00A02442DF7D
UUID_LogicalDevice	CBA00010-6C97-11D1-8271-00A02442DF7D
UUID_ACCO	CBA00040-6C97-11D1-8271-00A02442DF7D
UUID_RTAuto	CBA00050-6C97-11D1-8271-00A02442DF7D
UUID_SystemRTAuto	CBA00090-6C97-11D1-8271-00A02442DF7D

5.1.1.3.3 FAL ASE class interfaces

5.1.1.3.3.1 ICBAPhysicalDevice

```
[
    object,
    uuid(CBA00001-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPhysicalDevice Interface"),
    pointer_default(unique)
]
interface ICBAPhysicalDevice : IDispatch
{
    [propget, helpstring("Producer/manufacturer of the physical device")]
        HRESULT Producer([out, retval] BSTR *pVal);

    [propget, helpstring("Catalog name of the physical device")]
        HRESULT Product([out, retval] BSTR *pVal);

    [propget, helpstring("Serial number of the physical device")]
```

```
        HRESULT SerialNo([out, retval] VARIANT *pVal);

    [propget, helpstring("Production date of the physical device")]
        HRESULT ProductionDate([out, retval] DATE *pVal);

    [helpstring("Revision major.minor number of physical device")]
        HRESULT Revision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor);

    [propget, helpstring("Return logical device by name")]
        HRESULT LogicalDevice(
            [in]           BSTR Name,
            [out, retval] ICBALogicalDevice **ppLDev);
};
```

5.1.1.3.3.2 ICBAPhysicalDevice2

```
[
    object,
    uuid(CBA00006-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPhysicalDevice2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAPhysicalDevice2 : ICBAPhysicalDevice
{
    [helpstring("Implementation type")]
        HRESULT Type(
            [out] VARIANT_BOOL *pMultiApp,
            [out] VARIANT_BOOL *pPROFInetDCOMStack);

    [helpstring("Revision of PROFINET runtime source")]
        HRESULT PROFInetRevision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor,
            [out] SHORT *pServicePack,
            [out] SHORT *pBuild);

    [propget, helpstring("Return PDev stamp")]
        HRESULT PDevStamp([out, retval] LONG *pVal);
};
```

5.1.1.3.3.3 ICBABrowse

```
[
    object,
    uuid(CBA00002-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBABrowse Interface"),
    pointer_default(unique)
]
interface ICBABrowse : IDispatch
{
    [propget, helpstring("Return number of items")]
        HRESULT Count([out, retval] LONG *pVal);

    [helpstring("Browse all available items")]
        HRESULT BrowseItems(
            [in]               LONG     Offset,     // offset - !=0 if last call did
                                                     // not return all items
            [in]               LONG     MaxReturn, // at most return that many items
            [out]              VARIANT *pItem,      // SAFEARRAY(BSTR)  - item names
            [out, optional] VARIANT *pDataType,
            [out, optional] VARIANT *pAccessRight);

    // BrowseItems usage:
    //
    // on PhysicalDevice object:
    //    pItem - names of the LogicalDevices
    //    pDataType and pAccessRight - empty
    //
    // on LogicalDevice object:
    //    pItem - names of the non-system RTAutos
```

```
    //    pDataType and pAccessRight - empty
    //
    // on RTAuto object:
    //    pItem - names of the items
    //    pDataType - SAFEARRAY of LONG (actually VARTYPE), containing the
    //        stage 1 type description of the item
    //    pAccessRight - SAFEARRAY of LONG (actually ACCESSRIGHTSDEF), con-
    //       taining the access rights of the item
};
```

5.1.1.3.3.4 ICBABrowse2

```
[
    object,
    uuid(CBA00007-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBABrowse2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBABrowse2 : ICBABrowse
{
    [propget, helpstring("Return number of items")]
       HRESULT Count2(
           [in]               LONG      Selector,  // selects things to count
           [out, retval]      LONG     *pVal);      // count

    [helpstring("Browse all available items")]
       HRESULT BrowseItems2(
           [in]               LONG      Selector,  // selects things to browse
           [in]               LONG      Offset,    // offset - !=0 if last call did
                                                    // not return all items
           [in]               LONG      MaxReturn, // at most return that many items
           [out]              VARIANT *pItem,      // SAFEARRAY(BSTR)  - item names
           [out, optional] VARIANT *pInfo1,        // Additional info 1
           [out, optional] VARIANT *pInfo2);        // Additional info 2

    // BrowseItems2 usage:
    //
    // on PhysicalDevice object (Selector =0):
    //    pItem - names of the LogicalDevices
    //    pInfo1 and pInfo2 - empty
    //
    // on LogicalDevice object (Selector =0):
    //    pItem - names of the non-system RTAutos
    //    pInfo1 and pInfo2 - empty
    //
    // on LogicalDevice object (Selector =1):
    //    pItem - names of the system RTAutos (starting with "!")
    //    pInfo1 and pInfo2 - empty
    //
    // on RTAuto object (Selector =0):
    //    pItem - names of the items
    //    pInfo1 - SAFEARRAY of BSTR (actually, the BSTR must be interpreted
    //       as an ARRAY of SHORT), containing the stage 2 type description of
    //       the item
    //    pInfo2 - SAFEARRAY of LONG (actually ACCESSRIGHTSDEF), containing
    //       the access rights of the item
};
```

5.1.1.3.3.5 ICBAPersist

```
[
    object,
    uuid(CBA00005-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPersist Interface"),
    pointer_default(unique)
]
interface ICBAPersist : IDispatch
{
    [helpstring("Save all persistent data")]
       HRESULT Save();
};
```

5.1.1.3.3.6 ICBAPersist2

```
[
    object,
    uuid(CBA00008-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAPersist2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAPersist2 : ICBAPersist
{
    [helpstring("Save all persistent data")]
        HRESULT Save2(
            [out] VARIANT *pLDevName,         // SAFEARRAY(BSTR) - LDev names
            [out] VARIANT *pPartialResult);    // SAFEARRAY(LONG) - HRESULT partial
                                               //                    result
};
```

5.1.1.3.3.7 ICBALogicalDevice

```
[
    object,
    uuid(CBA00011-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBALogicalDevice Interface"),
    pointer_default(unique)
]
interface ICBALogicalDevice : IDispatch
{
    [propget, helpstring("Name of the logical device")]
        HRESULT Name([out, retval] BSTR *pVal);

    [propget, helpstring("Producer/manufacturer of the logical device")]
        HRESULT Producer([out, retval] BSTR *pVal);

    [propget, helpstring("Catalog name of the logical device")]
        HRESULT Product([out, retval] BSTR *pVal);

    [propget, helpstring("Serial number of the logical device")]
        HRESULT SerialNo([out, retval] VARIANT *pVal);

    [propget, helpstring("Production date of the logical device")]
        HRESULT ProductionDate([out, retval] DATE *pVal);

    [helpstring("Revision major.minor number of logical device")]
        HRESULT Revision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor);

    [propget, helpstring("Return pointer to ACCO")]
        HRESULT ACCO([out, retval] IUnknown **ppACCO);

    [propget, helpstring("Return RTAuto by name")]
        HRESULT RTAuto(
            [in]           BSTR Name,
            [out, retval] ICBARTAuto **ppAuto);
};
```

5.1.1.3.3.8 ICBALogicalDevice2

```
[
    object,
    uuid(CBA00017-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBALogicalDevice2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBALogicalDevice2 : ICBALogicalDevice
{
    [helpstring("Revision of PROFINET runtime source")]
        HRESULT PROFInetRevision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor,
            [out] SHORT *pServicePack,
```

```
            [out] SHORT *pBuild);

    [helpstring("Component information")]
        HRESULT ComponentInfo(
            [out] BSTR *pComponentID, // GUID as
                                      // "{7005c200-6C97-11D1-8271-00A02442DF7D}"
            [out] BSTR *pVersion);    // version
};
```

5.1.1.3.3.9 ICBAState

```
[
    object,
    uuid(CBA00012-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAState Interface"),
    pointer_default(unique)
]
interface ICBAState : IDispatch
{
    [propget, helpstring("State of the device")]
        HRESULT State([out, retval] STATEDEF *pVal);

    [helpstring("Activate logical device - transfer to state CBAOperating")]
        HRESULT Activate();

    [helpstring("Deactivate logical device - transfer to state CBAReady")]
        HRESULT Deactivate();

    [helpstring("Reset logical device - transfer via state CBAInitializing to
state CBAReady")]
        HRESULT Reset();

    [helpstring("Advise of the ICBAStateEvent")]
        HRESULT AdviseState(
            [in]  ICBAStateEvent *pStateEvent,
            [out] LONG *pCookie);

    [helpstring("Unadvise of the ICBAStateEvent")]
        HRESULT UnadviseState([in] LONG Cookie);
};
```

5.1.1.3.3.10 ICBATime

```
[
    object,
    uuid(CBA00014-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBATime Interface"),
    pointer_default(unique)
]
interface ICBATime : IDispatch
{
    [propget, helpstring("Get current time")]
        HRESULT Time([out, retval] DATE *pVal);

    [propput, helpstring("Set time")]
        HRESULT Time([in] DATE NewVal);
};
```

5.1.1.3.3.11 ICBAGroupError

```
[
    object,
    uuid(CBA00015-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBAGroupError Interface"),
    pointer_default(unique)
]
interface ICBAGroupError : IDispatch
{
    [helpstring("Get current error state")]
        HRESULT GroupError(
            [out] GROUPERRORDEF *pVal,
            [out] LONG *pMagicCookie);
```

```
    [helpstring("Advise of the ICBAGroupErrorEvent")]
        HRESULT AdviseGroupError(
            [in]  ICBAGroupErrorEvent *pGroupErrorEvent,
            [out] LONG *pCookie);

    [helpstring("Unadvise of the ICBAGroupErrorEvent")]
        HRESULT UnadviseGroupError([in] LONG Cookie);
};
```

5.1.1.3.3.12 ICBAAccoMgt

```
[
    object,
    uuid(CBA00041-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoMgt Interface"),
    pointer_default(unique)
]
interface ICBAAccoMgt : IUnknown
{
    [helpstring("Add a set of connections")]
        HRESULT AddConnections(
            [in, string] LPWSTR Provider,      // provider name ("PDev!LDev")
            [in]         WORD QoSType,         // common quality of service type
            [in]         WORD QoSValue,        // common quality of service value
            [in]         BOOLEAN State,        // activation state (active = TRUE)
            [in]         DWORD Count,          // number of connections
            [in, size_is(Count)] ADDCONNECTIONIN *pAddConnectionIn,
            // connection information
            [out, size_is(,Count)] ADDCONNECTIONOUT **ppAddConnectionOut);
            // results

    [helpstring("Remove a set of connections")]
        HRESULT RemoveConnections(
            [in]         DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pConsumerID,   // consumer IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Clear all connections")]
        HRESULT ClearConnections();

    [helpstring("Modify activation state of a set of connections")]
        HRESULT SetActivationState(
            [in]         BOOLEAN State,                // activation state
                                                        // (active = TRUE)
            [in]         DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pConsumerID,   // consumer IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Get resource information")]
        HRESULT GetInfo(
            [out]        DWORD *pMax,          // max. number of connections
            [out]        DWORD *pCurCnt);       // number of connections

    [helpstring("Get short diagnosis information")]
        HRESULT GetIDs(
            [out]        DWORD *pCount,                // number of connections
            [out, size_is(,*pCount)] GETIDOUT **ppGetIDOut); // short info

    [helpstring("Get full diagnosis information")]
        HRESULT GetConnections(
            [in]         DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pConsumerID,   // consumer IDs
            [out, size_is(,Count)] GETCONNECTIONOUT **ppGetConnectionOut); // full
info

    [helpstring("Revise quality of service")]
        HRESULT ReviseQoS(
            [in, string] LPWSTR RTAuto,                // RTAuto name
            [in]         WORD QoSType,                 // quality of service type
            [in]         WORD QoSValue,                // quality of service value
            [out]        WORD *pRevisedQoSValue);       // revised quality of
                                                        // service value
```

```
    [propget, helpstring("Get current ping factor")]
        HRESULT PingFactor([out, retval] WORD *pVal);

    [propput, helpstring("Set ping factor")]
        HRESULT PingFactor([in] WORD NewVal);

    [propget, helpstring("Return configuration data base cookie")]
        HRESULT CDBCookie([out, retval] LONG *pVal);
};
```

5.1.1.3.3.13 ICBAAccoMgt2

```
[
    object,
    uuid(CBA00046-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoMgt2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAAccoMgt2 : ICBAAccoMgt
{
    [helpstring("Get current consumer connection IDs")]
        HRESULT GetConsIDs(
            [out]          DWORD *pCount,                // number of connections
            [out, size_is(,*pCount)] DWORD **ppConsumerID);   // consumer IDs

    [helpstring("Get consumer connection information")]
        HRESULT GetConsConnections(
            [in]           DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pConsumerID, // consumer IDs
            [out, size_is(,Count)] GETCONSCONNOUT **ppGetConsConnOut);
                                                          // connection info

    [helpstring("Get consumer connection diagnosis")]
        HRESULT DiagConsConnections(
            [in]           DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pConsumerID, // consumer IDs
            [out, size_is(,Count)] DIAGCONSCONNOUT **ppDiagConsConnOut);
                                                          // diagnosis info

    [helpstring("Get current provider IDs")]
        HRESULT GetProvIDs(
            [out]          DWORD *pCount,                // number of connections
            [out, size_is(,*pCount)] DWORD **ppProviderID);   // provider IDs

    [helpstring("Get provider connection information")]
        HRESULT GetProvConnections(
            [in]           DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pProviderID, // provider IDs
            [out, size_is(,Count)] GETPROVCONNOUT **ppGetProvConnOut);
                                                          // connection info

    [helpstring("Get ACCO diagnosis")]
        HRESULT GetDiagnosis(
            [in]           DWORD Request,                // request
            [in]           DWORD InLength,               // length of InBuffer
            [in, size_is(InLength)] BYTE *pInBuffer, // InBuffer
            [out]          DWORD *pOutLength,            // length of OutBuffer
            [out, size_is(,*pOutLength)] BYTE **ppOutBuffer);  // OutBuffer
};
```

5.1.1.3.3.14 ICBAAccoCallback

```
[
    object,
    uuid(CBA00042-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoCallback Interface"),
    pointer_default(unique)
]
interface ICBAAccoCallback : IUnknown
{
    [helpstring("Hand over new connection data")]
        HRESULT OnDataChanged(
            [in]           long Length,                        // buffer length
            [in, size_is(Length)] unsigned char *pBuffer); // buffer
```

```
    // structure of pBuffer[]:
    //
    // header:
    //   00 UCHAR      version      -- constant=01
    //   01 UCHAR      flags
    //   02 USHORT     cnt          -- number of records
    //   04 array of record
    //
    // each record:
    //   00 USHORT     len          -- length of record
    //   02 ULONG      ConsumerID   -- consumer side of connection
    //   06 UCHAR      QualityCode  -- OPC QualityCode
    //   07            value        -- according to data type
    // len-7                        -- optional TimeStamp
    //
    // case ARRAY:
    //   07 USHORT     dim          -- array dimensions
    //   09 ULONG[dim] elem         -- number of elements in each dimension
    // 4*dim+09 etype[elem]         -- elements
    //
    // case BSTR:
    //   07 ULONG      strlen       -- string length (number of wide chars)
    //   11 USHORT[strlen] str      -- wide char string (not terminated with 0)
    //
    // case STRUCT: (stage 2)
    //   07 sequence of values, unaligned, depth-first
    //
    // in ARRAY or STRUCT:
    // sequence of values, unaligned, depth-first
    // representation: as above, with the following exceptions:
    // ARRAY: dimensions and number of elements fields are not present
    // BSTR: is filled with zeroes up to the maximum length for fixed layout
};
```

5.1.1.3.3.15 ICBAAccoCallback2

```
[
    object,
    uuid(CBA00047-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoCallback2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAAccoCallback2 : ICBAAccoCallback
{
    [helpstring("Reverse Ping from provider to consumer")]
        HRESULT Gnip();
};
```

5.1.1.3.3.16 ICBAAccoServer

```
[
    object,
    uuid(CBA00043-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoServer Interface"),
    pointer_default(unique)
]
interface ICBAAccoServer : IUnknown
{
    [helpstring("Establish a set of connections")]
        HRESULT Connect(
            [in, string] LPWSTR Consumer,      // consumer name ("PDev!LDev")
            [in]         WORD QoSType,         // common quality of service type
            [in]         WORD QoSValue,        // common quality of service value
            [in]         BOOLEAN State,        // activation state (active = TRUE)
            [in]         ICBAAccoCallback *pICBAAccoCallback, // interface pointer
                                                // to callback of the consumer
            [in]         DWORD Count,          // number of connections
            [in, size_is(Count)] CONNECTIN *pConnectIn, // connection information
            [out]        BOOLEAN *pFirstConnect,        // first connect call
            [out, size_is(,Count)] CONNECTOUT **ppConnectOut); // results

    [helpstring("Purge a set of connections")]
        HRESULT Disconnect(
            [in]         DWORD Count,                  // number of connections
```

```
            [in, size_is(Count)] DWORD *pProviderID,  // provider IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Purge all connections of a specific consumer")]
        HRESULT DisconnectMe([in, string] LPWSTR Consumer);
                                            // consumer name ("PDev!LDev")

    [helpstring("Modify activation state of a set of connections")]
        HRESULT SetActivation(
            [in]         BOOLEAN State,       // activation state (active = TRUE)
            [in]         DWORD Count,         // number of connections
            [in, size_is(Count)] DWORD *pProviderID,  // provider IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Lifebeat handling")]
        HRESULT Ping([in, string] LPWSTR Consumer); // consumer name ("PDev!LDev")
};
```

5.1.1.3.3.17 ICBAAccoServer2

```
[
    object,
    uuid(CBA00048-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoServer2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBAAccoServer2 : ICBAAccoServer
{
    [helpstring("Establish a set of connections")]
        HRESULT Connect2(
            [in, string] LPWSTR Consumer,   // consumer name ("PDev!LDev")
            [in]         WORD QoSType,      // common quality of service type
            [in]         WORD QoSValue,     // common quality of service value
            [in]         BOOLEAN State,     // activation state (active = TRUE)
            [in]         ICBAAccoCallback2 *pICBAAccoCallback, // interface
                                             // pointer to callback of the consumer
            [in]         DWORD Count,       // number of connections
            [in, size_is(Count)] CONNECTIN2 *pConnectIn, // connection information
            [out]        BOOLEAN *pFirstConnect,        // first connect call
            [out, size_is(,Count)] CONNECTOUT **ppConnectOut); // results

    [helpstring("Get the connection data")]
        HRESULT GetConnectionData(
            [in, string] LPWSTR Consumer,   // consumer name ("PDev!LDev")
            [out]        long *pLength,     // buffer length
            [out, size_is(,*pLength)] unsigned char **ppBuffer); // buffer
    // structure of pBuffer[]:
    //       see ICBAAccoCallback::OnDataChanged()
};
```

5.1.1.3.3.18 ICBAAccoServerSRT

```
[
    object,
    uuid(CBA00045-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoServerSRT Interface"),
    pointer_default(unique)
]
// stage 2 interface
interface ICBAAccoServerSRT : IUnknown
{
    [helpstring("Connect a set of communication relations")]
        HRESULT ConnectCR(
            [in, string] LPWSTR Consumer,         // consumer name ("PDev!LDev")
            [in]         WORD QoSType,            // RT type
            [in]         WORD QoSValue,           // value of RT
            [in]         ICBAAccoCallback2 *pICBAAccoCallback2, // interface
                                                   // pointer to the consumer's
                                                   // call-back to allow gnipping.
            [in]         MACAddr ConsumerMAC,     // MAC address of consumer
            [in]         DWORD Flags,             // flags
            [in]         DWORD Count,             // number of CRs to connect
            [in, size_is(Count)] CONNECTINCR *pConnectIn,       // CR information
            [out]        BOOLEAN *pFirstConnect,  // if true, AR was established
```

```
                                                  // with this call
            [out]        MACAddr *pProviderMAC,   // MAC address of provider
            [out, size_is(,Count)] CONNECTOUTCR **ppConnectOut); // CR result

    [helpstring("Disconnect a communication relation")]
        HRESULT DisconnectCR(
            [in]         DWORD Count,             // number of CRs to disconnect
            [in, size_is(Count)] DWORD *pProviderCRID, // provider CR IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Establish a set of connections")]
        HRESULT Connect(
            [in]         DWORD ProviderCRID,   // provider CR ID
            [in]         BOOLEAN State,        // activation state (active = TRUE)
            [in]         BOOLEAN LastConnect,  // CR is complete, start to
                                               // provide now
            [in]         DWORD Count,          // count of items to connect
            [in, size_is(Count)] CONNECTINSRT *pConnectIn,
                                               // connection information
            [out, size_is(,Count)] CONNECTOUT **ppConnectOut);
                                               // connection result

    [helpstring("Purge a set of connections")]
        HRESULT Disconnect(
            [in]         DWORD Count,                  // number of connections
            [in, size_is(Count)] DWORD *pProviderID,   // provider IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results

    [helpstring("Purge all connections of a specific consumer")]
        HRESULT DisconnectMe([in, string] LPWSTR Consumer);
                                               // consumer name ("PDev!LDev")

    [helpstring("Modify activation state of a set of connections")]
        HRESULT SetActivation(
            [in]         BOOLEAN State,        // activation state (active = TRUE)
            [in]         DWORD Count,          // number of connections
            [in, size_is(Count)] DWORD *pProviderID,   // provider IDs
            [out, size_is(,Count)] HRESULT **ppError); // partial results
}
```

5.1.1.3.3.19 ICBAAccoSync

```
[
    object,
    uuid(CBA00044-6C97-11D1-8271-00A02442DF7D),
    helpstring("ICBAAccoSync Interface"),
    pointer_default(unique)
]
interface ICBAAccoSync : IUnknown
{
    [helpstring("Read items")]
        HRESULT ReadItems(
            [in]         DWORD Count,                  // number of items to read
            [in, size_is(Count), string] LPWSTR *pReadItem, // items to read
            [out, size_is(,Count)] READITEMOUT **ppReadItemOut);
                                                       // item information

    [helpstring("Write items")]
        HRESULT WriteItems(
            [in]         DWORD Count,                  // number of items to write
            [in, size_is(Count)] WRITEITEMIN *pWriteItemIn, // items to write
            [out, size_is(,Count)] HRESULT **ppError);      // partial results

    [helpstring("Write items with quality code and time stamps")]
        HRESULT WriteItemsQCD(
            [in]         DWORD Count,                  // number of items to write
            [in, size_is(Count)] WRITEITEMQCDIN *pWriteItemQcdIn,
                                                       // items to write
            [out, size_is(,Count)] HRESULT **ppError); // partial results
};
```

5.1.1.3.3.20 ICBARTAuto

```
[
    object,
```

```
    uuid(CBA00051-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBARTAuto Interface"),
    pointer_default(unique)
]
interface ICBARTAuto : IDispatch
{
    [propget, helpstring("Name of the runtime automation object")]
        HRESULT Name([out, retval] BSTR *pVal);

    [helpstring("Revision major.minor number of the runtime automation object")]
        HRESULT Revision(
            [out] SHORT *pMajor,
            [out] SHORT *pMinor);
};
```

5.1.1.3.3.21 ICBARTAuto2

```
[
    object,
    uuid(CBA00052-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBARTAuto2 Interface"),
    pointer_default(unique)
]
// stage 2 extended interface
interface ICBARTAuto2 : ICBARTAuto
{
    [helpstring("Component information")]
        HRESULT ComponentInfo(
            [out] BSTR *pComponentID,  // GUID as
                                       // "{7005c200-6C97-11D1-8271-00A02442DF7D}"
            [out] BSTR *pVersion);     // version
};
```

5.1.1.3.3.22 ICBASystemProperties

```
[
    object,
    uuid(CBA00062-6C97-11D1-8271-00A02442DF7D),
    dual, oleautomation,
    helpstring("ICBASystemProperties Interface"),
    pointer_default(unique)
]
// stage 2 interface
interface ICBASystemProperties : IDispatch
{
    [propget, helpstring("State collection")]
        HRESULT StateCollection([out, retval] SAFEARRAY(BYTE) *pVal);
    // the SAFEARRAY contains:
    // extended type description (VT_USERDEFINED,5,VT_UI2,VT_UI2,VT_UI2,VT_UI2,
    // VT_UI2)
    // typedef struct STATECOLLECTION
    // {
    //     WORD  LDevState;              LDev's operating state (STATEDEF)
    //     WORD  LDevGroupError;         LDev's group error (GROUPERRORDEF)
    //     WORD  LDevGECookie;           LDev's group error cookie
    //     WORD  AccoGroupError;         Acco's group error (GROUPERRORDEF)
    //     WORD  AccoGECookie;           Acco's group error cookie
    // } STATECOLLECTION;

    [propget, helpstring("Stamp collection")]
        HRESULT StampCollection([out, retval] SAFEARRAY(BYTE) *pVal);
    // the SAFEARRAY contains:
    // extended type description (VT_USERDEFINED,3,VT_UI4,VT_UI4,VT_UI4)
    // typedef struct STAMPCOLLECTION
    // {
    //     DWORD  LDevStamp;             LDev stamp
    //     DWORD  AccoStampCDB;          ACCO stamp: persistent and volatile
    //                                               changes to CDB
    //     DWORD  AccoStampModCounter;   ACCO stamp: modification counter
    // } STAMPCOLLECTION;
};
```

5.1.1.3.4 FAL ASE classes

5.1.1.3.4.1 CBAPhysicalDevice

```
[
    uuid(CBA00000-6C97-11D1-8271-00A02442DF7D),
    helpstring("Physical device class")
]
coclass CBAPhysicalDevice
{
    [default]    interface ICBAPhysicalDevice;
                 interface ICBAPhysicalDevice2;
                 interface ICBABrowse;
                 interface ICBABrowse2;
                 interface ICBAPersist;
                 interface ICBAPersist2;
};
```

5.1.1.3.4.2 CBALogicalDevice

```
[
    uuid(CBA00010-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("Logical device class")
]
coclass CBALogicalDevice
{
    [default]    interface ICBALogicalDevice;
                 interface ICBALogicalDevice2;
                 interface ICBAState;
                 interface ICBATime;
                 interface ICBABrowse;
                 interface ICBABrowse2;
                 interface ICBAGroupError;
                 interface ICBAPersist;          //  only used by PC implementation
};
```

5.1.1.3.4.3 CBAAcco

```
[
    uuid(CBA00040-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("ACCO class")
]
coclass CBAAcco
{
    [default]    interface ICBAAccoMgt;
                 interface ICBAAccoMgt2;
                 interface ICBAAccoServer;
                 interface ICBAAccoServer2;
                 interface ICBAAccoServerSRT;
                 interface ICBAAccoCallback;
                 interface ICBAAccoCallback2;
                 interface ICBAAccoSync;
                 interface ICBAGroupError;
};
```

5.1.1.3.4.4 CBARTAuto

```
[
    uuid(CBA00050-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("RTAuto class")
]
coclass CBARTAuto
{
    [default]    interface ICBARTAuto;
                 interface ICBARTAuto2;
                 interface ICBABrowse;
                 interface ICBABrowse2;
    // plus additional interface(s) for items
    // or ICBASystemProperties for special "!SYSTEM" RTAuto
};
```

5.1.1.3.4.5 CBASystemRTAuto

```
[
```

```
    uuid(CBA00090-6C97-11D1-8271-00A02442DF7D),
    noncreatable,
    helpstring("System RTAuto class")
]
coclass CBASystemRTAuto
{
    [default]   interface ICBARTAuto;
                interface ICBARTAuto2;
                interface ICBABrowse;
                interface ICBABrowse2;
                interface ICBASystemProperties;
};
```

5.2 Transfer syntax

5.2.1 General

The transfer syntax is specified by the underlying ORPC wire protocol. The IDL defined by this specification shall be the basis for the APDU encoding and therefore the input for the ORPC model.

5.2.2 Serialization of values and parameter

5.2.2.1 General

Table 209 defines the serialization of the data types that are permitted for connections.

Table 209 – Data format for serialized connection data

Vartype	Data Type	Used Bytes in Data (ORPC channel)	Used Bytes in Data (RT channel)
VT_BOOL	2-byte signed int (True=-1, False=0)	2 bytes	
VT_I1	char	1 byte	
VT_UI1	unsigned char	1 byte	
VT_I2	short	2 bytes	
VT_UI2	unsigned short	2 bytes	
VT_I4	long	4 bytes	
VT_UI4	unsigned long	4 bytes	
VT_R4	float	4 bytes	
VT_R8	double	8 bytes	
VT_DATE	date, 8-byte real	8 bytes	
VT_BSTR	binary string	Byte count (VT_UI4) Afterwards (byte count / 2) * 2-byte wide chars	Byte count (VT_UI4) Afterwards (maximum byte count / 2) * 2-byte wide chars The maximum byte count of the consumer is used, even if the provider has a shorter BSTR, since the consumer assesses the layout of the buffer. Only the wide characters within the actual length of the BSTR are copied to the buffer, there is no need to null out the wide charac-ters between actual length and maximum length; they are treated as don't care.
VT_SAFEARRAY	multidimensional array	Array dimension (VT_UI2) for each dimension number of elements (VT_UI4) Serialized elements of the basis data type according to their declaration. The array elements are (like in C/C++) serialized line by line. This means that the last index runs the fastest.	Array dimension and number of elements in each dimension is omitted. Serialized elements of the basis data type ac-cording to their declaration. The array ele-ments (like in C/C++) are serialized line by line. This means that the last index runs the fastest.
VT_USERDEFINED	structure	The components of the structure are packed in the sequence of their declaration (e.g. without padding bytes) and stored as defined above.	

NOTE For an array the dimension and lengths will be omitted in the RT channel. The data type is clarified in the Connect service of the Acco Server SRT interface by transmitting the extended type description.

Arrays of BSTR and BSTR embedded in structures will always be transmitted with their maximum length; this applies to ORPC and RT channel.

5.2.2.2 Restrictions for the ORPC channel

Since a properties maximum connection data size is restricted to 32 kByte in the ORPC channel, only a maximum amount of data can be transmitted in a connection. This restricts the following data types, if they are used within a connection:

- Strings can have up to 16 382 wide characters.
- Arrays (one dimensional assumed) can have a maximum amount of data with 32 762 Byte.
- Structures are limited to a maximum size of 32 768 Byte.

A device may have smaller maximum sizes of its properties.

5.2.2.3 Restrictions for the RT channel

Since the maximum length of an item in the format defined above is restricted to 450 bytes in the RT channel and consistency of values is not ensured across multiple AccoDataCRs, the following data types are restricted within their length, if they are used within a connection:

- Strings can have up to 223 wide characters, since the actual byte count (4 bytes) is transmitted within the 450 bytes.
- Arrays (one-dimensional assumed) can have a maximum amount of data with 450 byte.
- Structures are limited to a maximum size of 450 byte.

NOTE Neither header nor item header are included in the maximum item size of 450 byte.

NOTE The maximum item size can have device specific further restrictions.

The size of the RT reference data is computed according to Table 210. It includes both overall header and all item headers. The maximum size of the RT reference data is 484 bytes, the minimum size is 40 bytes.

Table 210 – Calculation of the RT reference data size

Element	Needed length
Header	4 bytes
Per Item	For the Item Header 2 bytes
	For the Quality Code 1 byte
	For the data types VT_I1 and VT_UI1: 1 byte For the data types VT_BOOL, VT_I2 and VT_UI2: 2 bytes For the data types VT_I4, VT_UI4 and VT_R4: 4 bytes For the data type VT_R8: 8 bytes For the data type VT_BSTR: 4 bytes + maximum byte length For the data type VT_SAFEARRAY: Len_1 * ... * Len_{dim} * space per item For the data type VT_USERDEFINED: Space per item

5.3 FAL protocol state machines

The FAL protocol state machine structure is as defined in Figure 44. The general structure is according to the type 1 protocol machine model.

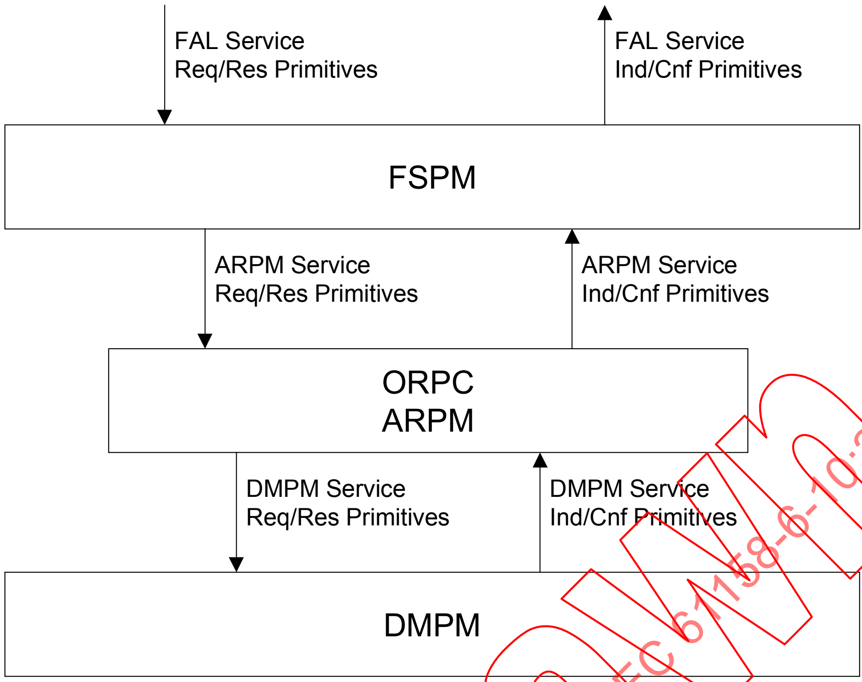

Figure 44 – Relationship among protocol machines

The behavior of the FAL is specified by three integrated protocol machines. The FSPM is the service interface between the FAL services that are part of the FAL Class specification and the particular AREP. Currently, the type 10 specifies exactly one AREP class and shall have only one instance of this class. Therefore, the common service attribute AREP or AREP_ID shall be omitted and an implicit AREP selection is used. The addressing of the FAL object interface instance is specified via the common class and service attribute Interface Pointer.

The FSPM is responsible for the following activities:

- To accept service primitives from the FAL service user and convert them into FAL internal primitives.
- To select the ORPC ARPM state machine based on the implicit addressing mechanism and send FAL internal primitives with unified service parameters to the ORPC ARPM.
- To accept FAL internal primitives from the ARPM and convert them into service primitives for the FAL service user.
- To deliver the FAL service primitives to the FAL user or object instances addressed via the service attribute Interface Pointer.

The ARPM specifies the conveyance type for the application relation and adds the reference to the appropriated IDL to support the data marshaling of the underlying ORPC model.

The DMPM specifies the mapping to an abstract ORPC model. The ORPC model defines therefore an abstract service interface.

5.4 AP context state machine

There is no AP Context State Machine defined for the FAL AE.

5.5 FAL service protocol machines (FSPM)

5.5.1 Overview

This type specifies one FSPM state machine to transfer the FAL user service primitives into FAL internal service primitives and vice versa.

5.5.2 Primitive definitions

5.5.2.1 Primitives exchanged between FSPM and FAL User

Table 211 shows the service primitives including their associated parameters issued by the FAL User and received by the FSPM.

Table 211 – Primitives issued by FAL User to FSPM

Primitive name	Source	Associated parameters	Functions
QueryInterface.req	FAL User	Interface Pointer, Interface ID	Refer to FAL Service Definition in IEC 61158-5-10
QueryInterface.rsp(+)	FAL User	Interface Pointer, hresult, ppvObject	
QueryInterface.rsp(-)	FAL User	Interface Pointer, hresult	
AddRef.req	FAL User	Interface Pointer	
AddRef.rsp	FAL User	Interface Pointer, Reference Count	
Release.req	FAL User	Interface Pointer	
Release.rsp	FAL User	Interface Pointer, Reference Count	
GetTypeInfoCount.req	FAL User	Interface Pointer	
GetTypeInfoCount.rsp(+)	FAL User	Interface Pointer, hresult, pcTInfo	
GetTypeInfoCount.rsp(-)	FAL User	Interface Pointer, hresult	
GetTypeInfo.req	FAL User	Interface Pointer, iTInfo, lcid	
GetTypeInfo.rsp(+)	FAL User	Interface Pointer, hresult, ppTInfo	
GetTypeInfo.rsp(-)	FAL User	Interface Pointer, hresult	
GetIDsOfNames.req	FAL User	Interface Pointer, riid, List of rgszNames, cNames, lcid	
GetIDsOfNames.rsp(+)	FAL User	Interface Pointer, hresult, List of rgDispId	
GetIDsOfNames.rsp(-)	FAL User	Interface Pointer, hresult	
Invoke.req	FAL User	Interface Pointer, dispIdMember, riid, lcid, wFlags, pDispParams	
Invoke.rsp(+)	FAL User	Interface Pointer, hresult, pVarResult	
Invoke.rsp(-)	FAL User	Interface Pointer, hresult, pExcepInfo, puArgErr	
get_Producer.req	FAL User	Interface Pointer	
get_Producer.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Producer.rsp(-)	FAL User	Interface Pointer, hresult	
get_Product.req	FAL User	Interface Pointer	
get_Product.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Product.rsp(-)	FAL User	Interface Pointer, hresult	
get_SerialNo.req	FAL User	Interface Pointer	
get_SerialNo.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_SerialNo.rsp(-)	FAL User	Interface Pointer, hresult	
get_ProductionDate.req	FAL User	Interface Pointer	

Primitive name	Source	Associated parameters	Functions
get_ProductionDate.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_ProductionDate.rsp(-)	FAL User	Interface Pointer, hresult	
Revision.req	FAL User	Interface Pointer	
Revision.rsp(+)	FAL User	Interface Pointer, hresult, pMajor, pMinor	
Revision.rsp(-)	FAL User	Interface Pointer, hresult	
get_LogicalDevice.req	FAL User	Interface Pointer, Name	
get_LogicalDevice.rsp(+)	FAL User	Interface Pointer, hresult, ppLDev	
get_LogicalDevice.rsp(-)	FAL User	Interface Pointer, hresult	
Type.req	FAL User	Interface Pointer	
Type.rsp(+)	FAL User	Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack	
Type.rsp(-)	FAL User	Interface Pointer, hresult	
PROFInetRevision.req	FAL User	Interface Pointer	
PROFInetRevision.rsp(+)	FAL User	Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild	
PROFInetRevision.rsp(-)	FAL User	Interface Pointer, hresult	
get_PDevStamp.req	FAL User	Interface Pointer	
get_PDevStamp.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_PDevStamp.rsp(-)	FAL User	Interface Pointer, hresult	
get_Count.req	FAL User	Interface Pointer	
get_Count.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Count.rsp(-)	FAL User	Interface Pointer, hresult	
BrowseItems.req	FAL User	Interface Pointer, Offset, MaxReturn	
BrowseItems.rsp(+)	FAL User	Interface Pointer, hresult, pItem, pDataType, pAccessRight	
BrowseItems.rsp(-)	FAL User	Interface Pointer, hresult	
get_Count2.req	FAL User	Interface Pointer, Selector	
get_Count2.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Count2.rsp(-)	FAL User	Interface Pointer, hresult	
BrowseItems2.req	FAL User	Interface Pointer, Selector, Offset, MaxReturn	
BrowseItems2.rsp(+)	FAL User	Interface Pointer, hresult, pItem, pInfo1, pInfo2	
BrowseItems2.rsp(-)	FAL User	Interface Pointer, hresult	
Save.req	FAL User	Interface Pointer	
Save.rsp(+)	FAL User	Interface Pointer, hresult	
Save.rsp(-)	FAL User	Interface Pointer, hresult	
Save2.req	FAL User	Interface Pointer	
Save2.rsp(+)	FAL User	Interface Pointer, hresult, pLDevName, pPartialResult	
Save2.rsp(-)	FAL User	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
get_Name.req	FAL User	Interface Pointer	
get_Name.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Name.rsp(-)	FAL User	Interface Pointer, hresult	
get_ACCO.req	FAL User	Interface Pointer	
get_ACCO.rsp(+)	FAL User	Interface Pointer, hresult, ppACCO	
get_ACCO.rsp(-)	FAL User	Interface Pointer, hresult	
get_RTAuto.req	FAL User	Interface Pointer, Name	
get_RTAuto.rsp(+)	FAL User	Interface Pointer, hresult, ppAuto	
get_RTAuto.rsp(-)	FAL User	Interface Pointer, hresult	
ComponentInfo.req	FAL User	Interface Pointer	
ComponentInfo.rsp(+)	FAL User	Interface Pointer, hresult, pComponentID, pVersion	
ComponentInfo.rsp(-)	FAL User	Interface Pointer, hresult	
get_State.req	FAL User	Interface Pointer	
get_State.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_State.rsp(-)	FAL User	Interface Pointer, hresult	
Activate.req	FAL User	Interface Pointer	
Activate.rsp(+)	FAL User	Interface Pointer, hresult	
Activate.rsp(-)	FAL User	Interface Pointer, hresult	
Deactivate.req	FAL User	Interface Pointer	
Deactivate.rsp(+)	FAL User	Interface Pointer, hresult	
Deactivate.rsp(-)	FAL User	Interface Pointer, hresult	
Reset.req	FAL User	Interface Pointer	
Reset.rsp(+)	FAL User	Interface Pointer, hresult	
Reset.rsp(-)	FAL User	Interface Pointer, hresult	
AdviseState.req	FAL User	Interface Pointer, pStateEvent	
AdviseState.rsp(+)	FAL User	Interface Pointer, hresult, pCookie	
AdviseState.rsp(-)	FAL User	Interface Pointer, hresult	
UnadviseState.req	FAL User	Interface Pointer, Cookie	
UnadviseState.rsp(+)	FAL User	Interface Pointer, hresult	
UnadviseState.rsp(-)	FAL User	Interface Pointer, hresult	
get_Time.req	FAL User	Interface Pointer	
get_Time.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_Time.rsp(-)	FAL User	Interface Pointer, hresult	
put_Time.req	FAL User	Interface Pointer, NewVal	
put_Time.rsp(+)	FAL User	Interface Pointer, hresult	
put_Time.rsp(-)	FAL User	Interface Pointer, hresult	
GroupError.req	FAL User	Interface Pointer	
GroupError.rsp(+)	FAL User	Interface Pointer, hresult, pVal, pMagicCookie	
GroupError.rsp(-)	FAL User	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
AdviseGroupError.req	FAL User	Interface Pointer, pGroupErrorEvent	
AdviseGroupError.rsp(+)	FAL User	Interface Pointer, hresult, pCookie	
AdviseGroupError.rsp(-)	FAL User	Interface Pointer, hresult	
UnAdviseGroupError.req	FAL User	Interface Pointer, Cookie	
UnAdviseGroupError.rsp(+)	FAL User	Interface Pointer, hresult	
UnAdviseGroupError.rsp(-)	FAL User	Interface Pointer, hresult	
AddConnections.req	FAL User	Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn	
AddConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppAddConnectionOut	
AddConnections.rsp(-)	FAL User	Interface Pointer, hresult	
RemoveConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
RemoveConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
RemoveConnections.rsp(-)	FAL User	Interface Pointer, hresult	
ClearConnections.req	FAL User	Interface Pointer	
ClearConnections.rsp(+)	FAL User	Interface Pointer, hresult	
ClearConnections.rsp(-)	FAL User	Interface Pointer, hresult	
SetActivationState.req	FAL User	Interface Pointer, State, Count, List of pConsumerID	
SetActivationState.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
SetActivationState.rsp(-)	FAL User	Interface Pointer, hresult	
GetInfo.req	FAL User	Interface Pointer	
GetInfo.rsp(+)	FAL User	Interface Pointer, hresult, pMax, pCurCnt	
GetInfo.rsp(-)	FAL User	Interface Pointer, hresult	
GetIDs.req	FAL User	Interface Pointer	
GetIDs.rsp(+)	FAL User	Interface Pointer, hresult, pCount, List of ppGetIDOut	
GetIDs.rsp(-)	FAL User	Interface Pointer, hresult	
GetConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
GetConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppGetConnectionOut	
GetConnections.rsp(-)	FAL User	Interface Pointer, hresult	
ReviseQoS.req	FAL User	Interface Pointer, RTAuto, QoSType, QoSValue	
ReviseQoS.rsp(+)	FAL User	Interface Pointer, hresult, pRevisedQoSValue	
ReviseQoS.rsp(-)	FAL User	Interface Pointer, hresult	
get_PingFactor.req	FAL User	Interface Pointer	
get_PingFactor.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_PingFactor.rsp(-)	FAL User	Interface Pointer, hresult	
put_PingFactor.req	FAL User	Interface Pointer, NewVal	

Primitive name	Source	Associated parameters	Functions
put_PingFactor.rsp(+)	FAL User	Interface Pointer, hresult	
put_PingFactor.rsp(-)	FAL User	Interface Pointer, hresult	
get_CDBCookie.req	FAL User	Interface Pointer	
get_CDBCookie.rsp(+)	FAL User	Interface Pointer, hresult, pVal	
get_CDBCookie.rsp(-)	FAL User	Interface Pointer, hresult	
GetConsIDs.req	FAL User	Interface Pointer	
GetConsIDs.rsp(+)	FAL User	Interface Pointer, hresult, pCount, List of ppConsumerID	
GetConsIDs.rsp(-)	FAL User	Interface Pointer, hresult	
GetConsConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
GetConsConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppGetConsConnOut	
GetConsConnections.rsp(-)	FAL User	Interface Pointer, hresult	
DiagConsConnections.req	FAL User	Interface Pointer, Count, List of pConsumerID	
DiagConsConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppDiagConsConnOut	
DiagConsConnections.rsp(-)	FAL User	Interface Pointer, hresult	
GetProvIDs.req	FAL User	Interface Pointer	
GetProvIDs.rsp(+)	FAL User	Interface Pointer, hresult, pCount, List of ppProviderID	
GetProvIDs.rsp(-)	FAL User	Interface Pointer, hresult	
GetProvConnections.req	FAL User	Interface Pointer, Count, List of pProviderID	
GetProvConnections.rsp(+)	FAL User	Interface Pointer, hresult, List of ppGetProvConnOut	
GetProvConnections.rsp(-)	FAL User	Interface Pointer, hresult	
GetDiagnosis.req	FAL User	Interface Pointer, Request, InLength, pInBuffer	
GetDiagnosis.rsp(+)	FAL User	Interface Pointer, hresult, pOutLength, ppOutBuffer	
GetDiagnosis.rsp(-)	FAL User	Interface Pointer, hresult	
Connect.req	FAL User	Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn	
Connect.rsp(+)	FAL User	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect.rsp(-)	FAL User	Interface Pointer, hresult	
Disconnect.req	FAL User	Interface Pointer, Count, List of pProviderID	
Disconnect.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
Disconnect.rsp(-)	FAL User	Interface Pointer, hresult	
DisconnectMe.req	FAL User	Interface Pointer, Consumer	
DisconnectMe.rsp(+)	FAL User	Interface Pointer, hresult	
DisconnectMe.rsp(-)	FAL User	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
SetActivation.req	FAL User	Interface Pointer, State, Count, List of pProviderID	
SetActivation.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
SetActivation.rsp(-)	FAL User	Interface Pointer, hresult	
Ping.req	FAL User	Interface Pointer, Consumer	
Ping.rsp(+)	FAL User	Interface Pointer, hresult	
Ping.rsp(-)	FAL User	Interface Pointer, hresult	
Connect2.req	FAL User	Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn	
Connect2.rsp(+)	FAL User	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect2.rsp(-)	FAL User	Interface Pointer, hresult	
GetConnectionData.req	FAL User	Interface Pointer, Consumer	
GetConnectionData.rsp(+)	FAL User	Interface Pointer, hresult, pLength, pBuffer	
GetConnectionData.rsp(-)	FAL User	Interface Pointer, hresult	
OnDataChanged.req	FAL User	Interface Pointer, Length, pBuffer	
OnDataChanged.rsp(+)	FAL User	Interface Pointer, hresult	
OnDataChanged.rsp(-)	FAL User	Interface Pointer, hresult	
Gnip.req	FAL User	Interface Pointer	
Gnip.rsp(+)	FAL User	Interface Pointer, hresult	
Gnip.rsp(-)	FAL User	Interface Pointer, hresult	
ReadItems.req	FAL User	Interface Pointer, Count, List of pReadItem	
ReadItems.rsp(+)	FAL User	Interface Pointer, hresult, List of ppReadItemOut	
ReadItems.rsp(-)	FAL User	Interface Pointer, hresult	
WriteItems.req	FAL User	Interface Pointer, Count, List of pWriteItem	
WriteItems.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
WriteItems.rsp(-)	FAL User	Interface Pointer, hresult	
WriteItemsQCD.req	FAL User	Interface Pointer, Count, List of pWriteItemQcdIn	
WriteItemsQCD.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
WriteItemsQCD.rsp(-)	FAL User	Interface Pointer, hresult	
ConnectCR.req	FAL User	Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn	
ConnectCR.rsp(+)	FAL User	Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut	
ConnectCR.rsp(-)	FAL User	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
DisconnectCR.req	FAL User	Interface Pointer, Count, List of pProviderCRID	
DisconnectCR.rsp(+)	FAL User	Interface Pointer, hresult, List of ppError	
DisconnectCR.rsp(-)	FAL User	Interface Pointer, hresult	
SRTConnect.req	FAL User	Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn	
SRTConnect.rsp(+)	FAL User	Interface Pointer, hresult, List of ppConnectOut	
SRTConnect.rsp(-)	FAL User	Interface Pointer, hresult	
CoCreateInstance.req	FAL User	Host address, Class ID, Interface ID,	
CoCreateInstance.rsp(+)	FAL User	hresult, Interface Pointer	
CoCreateInstance.rsp(-)	FAL User	hresult	
CoDisconnectObject.req	FAL User	Interface Pointer	
CoDisconnectObject.rsp	FAL User	Interface Pointer, hresult	
Call.req	FSPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	
Call.rsp(+)	FSPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	
Call.rsp(-)	FSPM	Interface Pointer, Service Name, hresult	

Table 212 shows the service primitives including their associated parameters issued by the FSPM and received by the FAL User.

Table 212 – Primitives issued by FSPM to FAL User

Primitive name	Source	Associated parameters	Functions
QueryInterface.ind	FSPM	Interface Pointer, Interface ID	Refer to FAL Service Definition in IEC 61158-5-10
QueryInterface.cnf(+)	FSPM	Interface Pointer, hresult, ppvObject	
QueryInterface.cnf(-)	FSPM	Interface Pointer, hresult	
AddRef.ind	FSPM	Interface Pointer	
AddRef.cnf	FSPM	Interface Pointer, Reference Count	
Release.ind	FSPM	Interface Pointer	
Release.cnf	FSPM	Interface Pointer, Reference Count	
GetTypeInfoCount.ind	FSPM	Interface Pointer	
GetTypeInfoCount.cnf(+)	FSPM	Interface Pointer, hresult, pcTInfo	
GetTypeInfoCount.cnf(-)	FSPM	Interface Pointer, hresult	
GetTypeInfo.ind	FSPM	Interface Pointer, iTInfo, lcid	
GetTypeInfo.cnf(+)	FSPM	Interface Pointer, hresult, ppTInfo	
GetTypeInfo.cnf(-)	FSPM	Interface Pointer, hresult	
GetIDsOfNames.ind	FSPM	Interface Pointer, riid, List of rgszNames, cNames, lcid	
GetIDsOfNames.cnf(+)	FSPM	Interface Pointer, hresult, List of rgDispId	
GetIDsOfNames.cnf(-)	FSPM	Interface Pointer, hresult	
Invoke.ind	FSPM	Interface Pointer, dispIdMember, riid, lcid, wFlags, pDispParams	
Invoke.cnf(+)	FSPM	Interface Pointer, hresult, pVarResult	
Invoke.cnf(-)	FSPM	Interface Pointer, hresult, pExcepInfo, puArgErr	
get_Producer.ind	FSPM	Interface Pointer	
get_Producer.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Producer.cnf(-)	FSPM	Interface Pointer, hresult	
get_Product.ind	FSPM	Interface Pointer	
get_Product.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Product.cnf(-)	FSPM	Interface Pointer, hresult	
get_SerialNo.ind	FSPM	Interface Pointer	
get_SerialNo.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_SerialNo.cnf(-)	FSPM	Interface Pointer, hresult	
get_ProductionDate.ind	FSPM	Interface Pointer	
get_ProductionDate.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_ProductionDate.cnf(-)	FSPM	Interface Pointer, hresult	
Revision.ind	FSPM	Interface Pointer	
Revision.cnf(+)	FSPM	Interface Pointer, hresult, pMajor, pMinor	

Primitive name	Source	Associated parameters	Functions
Revision.cnf(-)	FSPM	Interface Pointer, hresult	
get_LogicalDevice.ind	FSPM	Interface Pointer, Name	
get_LogicalDevice.cnf(+)	FSPM	Interface Pointer, hresult, ppLDev	
get_LogicalDevice.cnf(-)	FSPM	Interface Pointer, hresult	
Type.ind	FSPM	Interface Pointer	
Type.cnf(+)	FSPM	Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack	
Type.cnf(-)	FSPM	Interface Pointer, hresult	
PROFInetRevision.ind	FSPM	Interface Pointer	
PROFInetRevision.cnf(+)	FSPM	Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild	
PROFInetRevision.cnf(-)	FSPM	Interface Pointer, hresult	
get_PDevStamp.ind	FSPM	Interface Pointer	
get_PDevStamp.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_PDevStamp.cnf(-)	FSPM	Interface Pointer, hresult	
get_Count.ind	FSPM	Interface Pointer	
get_Count.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Count.cnf(-)	FSPM	Interface Pointer, hresult	
BrowseItems.ind	FSPM	Interface Pointer, Offset, MaxReturn	
BrowseItems.cnf(+)	FSPM	Interface Pointer, hresult, pItem, pDataType, pAccessRight	
BrowseItems.cnf(-)	FSPM	Interface Pointer, hresult	
get_Count2.ind	FSPM	Interface Pointer, Selector	
get_Count2.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Count2.cnf(-)	FSPM	Interface Pointer, hresult	
BrowseItems2.ind	FSPM	Interface Pointer, Selector, Offset, MaxReturn	
BrowseItems2.cnf(+)	FSPM	Interface Pointer, hresult, pItem, pInfo1, pInfo2	
BrowseItems2.cnf(-)	FSPM	Interface Pointer, hresult	
Save.ind	FSPM	Interface Pointer	
Save.cnf(+)	FSPM	Interface Pointer, hresult	
Save.cnf(-)	FSPM	Interface Pointer, hresult	
Save2.ind	FSPM	Interface Pointer	
Save2.cnf(+)	FSPM	Interface Pointer, hresult, pLDevName, pPartialResult	
Save2.cnf(-)	FSPM	Interface Pointer, hresult	
get_Name.ind	FSPM	Interface Pointer	
get_Name.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Name.cnf(-)	FSPM	Interface Pointer, hresult	
get_ACCO.ind	FSPM	Interface Pointer	

Primitive name	Source	Associated parameters	Functions
get_ACCO.cnf(+)	FSPM	Interface Pointer, hresult, ppACCO	
get_ACCO.cnf(-)	FSPM	Interface Pointer, hresult	
get_RTAuto.ind	FSPM	Interface Pointer, Name	
get_RTAuto.cnf(+)	FSPM	Interface Pointer, hresult, ppAuto	
get_RTAuto.cnf(-)	FSPM	Interface Pointer, hresult	
ComponentInfo.ind	FSPM	Interface Pointer	
ComponentInfo.cnf(+)	FSPM	Interface Pointer, hresult, pComponentID, pVersion	
ComponentInfo.cnf(-)	FSPM	Interface Pointer, hresult	
get_State.ind	FSPM	Interface Pointer	
get_State.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_State.cnf(-)	FSPM	Interface Pointer, hresult	
Activate.ind	FSPM	Interface Pointer	
Activate.cnf(+)	FSPM	Interface Pointer, hresult	
Activate.cnf(-)	FSPM	Interface Pointer, hresult	
Deactivate.ind	FSPM	Interface Pointer	
Deactivate.cnf(+)	FSPM	Interface Pointer, hresult	
Deactivate.cnf(-)	FSPM	Interface Pointer, hresult	
Reset.ind	FSPM	Interface Pointer	
Reset.cnf(+)	FSPM	Interface Pointer, hresult	
Reset.cnf(-)	FSPM	Interface Pointer, hresult	
AdviseState.ind	FSPM	Interface Pointer, pStateEvent	
AdviseState.cnf(+)	FSPM	Interface Pointer, hresult, pCookie	
AdviseState.cnf(-)	FSPM	Interface Pointer, hresult	
UnadviseState.ind	FSPM	Interface Pointer, Cookie	
UnadviseState.cnf(+)	FSPM	Interface Pointer, hresult	
UnadviseState.cnf(-)	FSPM	Interface Pointer, hresult	
get_Time.ind	FSPM	Interface Pointer	
get_Time.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_Time.cnf(-)	FSPM	Interface Pointer, hresult	
put_Time.ind	FSPM	Interface Pointer, NewVal	
put_Time.cnf(+)	FSPM	Interface Pointer, hresult	
put_Time.cnf(-)	FSPM	Interface Pointer, hresult	
GroupError.ind	FSPM	Interface Pointer	
GroupError.cnf(+)	FSPM	Interface Pointer, hresult, pVal, pMagicCookie	
GroupError.cnf(-)	FSPM	Interface Pointer, hresult	
AdviseGroupError.ind	FSPM	Interface Pointer, pGroupErrorEvent	
AdviseGroupError.cnf(+)	FSPM	Interface Pointer, hresult, pCookie	
AdviseGroupError.cnf(-)	FSPM	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
UnAdviseGroupError.ind	FSPM	Interface Pointer, Cookie	
UnAdviseGroupError.cnf(+)	FSPM	Interface Pointer, hresult	
UnAdviseGroupError.cnf(-)	FSPM	Interface Pointer, hresult	
AddConnections.ind	FSPM	Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn	
AddConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppAddConnectionOut	
AddConnections.cnf(-)	FSPM	Interface Pointer, hresult	
RemoveConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
RemoveConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
RemoveConnections.cnf(-)	FSPM	Interface Pointer, hresult	
ClearConnections.ind	FSPM	Interface Pointer	
ClearConnections.cnf(+)	FSPM	Interface Pointer, hresult	
ClearConnections.cnf(-)	FSPM	Interface Pointer, hresult	
SetActivationState.ind	FSPM	Interface Pointer, State, Count, List of pConsumerID	
SetActivationState.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
SetActivationState.cnf(-)	FSPM	Interface Pointer, hresult	
GetInfo.ind	FSPM	Interface Pointer	
GetInfo.cnf(+)	FSPM	Interface Pointer, hresult, pMax, pCurCnt	
GetInfo.cnf(-)	FSPM	Interface Pointer, hresult	
GetIDs.ind	FSPM	Interface Pointer	
GetIDs.cnf(+)	FSPM	Interface Pointer, hresult, pCount, List of ppGetIDOut	
GetIDs.cnf(-)	FSPM	Interface Pointer, hresult	
GetConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
GetConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppGetConnectionOut	
GetConnections.cnf(-)	FSPM	Interface Pointer, hresult	
ReviseQoS.ind	FSPM	Interface Pointer, RTAuto, QoSType, QoSValue	
ReviseQoS.cnf(+)	FSPM	Interface Pointer, hresult, pRevisedQoSValue	
ReviseQoS.cnf(-)	FSPM	Interface Pointer, hresult	
get_PingFactor.ind	FSPM	Interface Pointer	
get_PingFactor.cnf(+)	FSPM	Interface Pointer, hresult, pVal	
get_PingFactor.cnf(-)	FSPM	Interface Pointer, hresult	
put_PingFactor.ind	FSPM	Interface Pointer, NewVal	
put_PingFactor.cnf(+)	FSPM	Interface Pointer, hresult	
put_PingFactor.cnf(-)	FSPM	Interface Pointer, hresult	
get_CDBCookie.ind	FSPM	Interface Pointer	
get_CDBCookie.cnf(+)	FSPM	Interface Pointer, hresult, pVal	

Primitive name	Source	Associated parameters	Functions
get_CDBCookie.cnf(-)	FSPM	Interface Pointer, hresult	
GetConsIDs.ind	FSPM	Interface Pointer	
GetConsIDs.cnf(+)	FSPM	Interface Pointer, hresult, pCount, List of ppConsumerID	
GetConsIDs.cnf(-)	FSPM	Interface Pointer, hresult	
GetConsConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
GetConsConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppGetConsConnOut	
GetConsConnections.cnf(-)	FSPM	Interface Pointer, hresult	
DiagConsConnections.ind	FSPM	Interface Pointer, Count, List of pConsumerID	
DiagConsConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppDiagConsConnOut	
DiagConsConnections.cnf(-)	FSPM	Interface Pointer, hresult	
GetProvIDs.ind	FSPM	Interface Pointer	
GetProvIDs.cnf(+)	FSPM	Interface Pointer, hresult, pCount, List of ppProviderID	
GetProvIDs.cnf(-)	FSPM	Interface Pointer, hresult	
GetProvConnections.ind	FSPM	Interface Pointer, Count, List of pProviderID	
GetProvConnections.cnf(+)	FSPM	Interface Pointer, hresult, List of ppGetProvConnOut	
GetProvConnections.cnf(-)	FSPM	Interface Pointer, hresult	
GetDiagnosis.ind	FSPM	Interface Pointer, Request, lnLength, pInBuffer	
GetDiagnosis.cnf(+)	FSPM	Interface Pointer, hresult, pOutLength, ppOutBuffer	
GetDiagnosis.cnf(-)	FSPM	Interface Pointer, hresult	
Connect.ind	FSPM	Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn	
Connect.cnf(+)	FSPM	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect.cnf(-)	FSPM	Interface Pointer, hresult	
Disconnect.ind	FSPM	Interface Pointer, Count, List of pProviderID	
Disconnect.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
Disconnect.cnf(-)	FSPM	Interface Pointer, hresult	
DisconnectMe.ind	FSPM	Interface Pointer, Consumer	
DisconnectMe.cnf(+)	FSPM	Interface Pointer, hresult	
DisconnectMe.cnf(-)	FSPM	Interface Pointer, hresult	
SetActivation.ind	FSPM	Interface Pointer, State, Count, List of pProviderID	
SetActivation.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
SetActivation.cnf(-)	FSPM	Interface Pointer, hresult	
Ping.ind	FSPM	Interface Pointer, Consumer	

Primitive name	Source	Associated parameters	Functions
Ping.cnf(+)	FSPM	Interface Pointer, hresult	
Ping.cnf(-)	FSPM	Interface Pointer, hresult	
Connect2.ind	FSPM	Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn	
Connect2.cnf(+)	FSPM	Interface Pointer, hresult, pFirstConnect, List of ppConnectOut	
Connect2.cnf(-)	FSPM	Interface Pointer, hresult	
OnDataChanged.ind	FSPM	Interface Pointer, Length, pBuffer	
GetConnectionData.ind	FSPM	Interface Pointer, Consumer	
GetConnectionData.cnf(+)	FSPM	Interface Pointer, hresult, pLength, pBuffer	
GetConnectionData.cnf(-)	FSPM	Interface Pointer, hresult	
OnDataChanged.cnf(+)	FSPM	Interface Pointer, hresult	
OnDataChanged.cnf(-)	FSPM	Interface Pointer, hresult	
Gnip.ind	FSPM	Interface Pointer	
Gnip.cnf(+)	FSPM	Interface Pointer, hresult	
Gnip.cnf(-)	FSPM	Interface Pointer, hresult	
ReadItems.ind	FSPM	Interface Pointer, Count, List of pReadItem	
ReadItems.cnf(+)	FSPM	Interface Pointer, hresult, List of ppReadItemOut	
ReadItems.cnf(-)	FSPM	Interface Pointer, hresult	
WriteItems.ind	FSPM	Interface Pointer, Count, List of pWriteItem	
WriteItems.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
WriteItems.cnf(-)	FSPM	Interface Pointer, hresult	
WriteItemsQCD.ind	FSPM	Interface Pointer, Count, List of pWriteItemQcdIn	
WriteItemsQCD.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
WriteItemsQCD.cnf(-)	FSPM	Interface Pointer, hresult	
ConnectCR.ind	FSPM	Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn	
ConnectCR.cnf(+)	FSPM	Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut	
ConnectCR.cnf(-)	FSPM	Interface Pointer, hresult	
DisconnectCR.ind	FSPM	Interface Pointer, Count, List of pProviderCRID	
DisconnectCR.cnf(+)	FSPM	Interface Pointer, hresult, List of ppError	
DisconnectCR.cnf(-)	FSPM	Interface Pointer, hresult	

Primitive name	Source	Associated parameters	Functions
SRTConnect.ind	FSPM	Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn	
SRTConnect.cnf(+)	FSPM	Interface Pointer, hresult, List of ppConnectOut	
SRTConnect.cnf(-)	FSPM	Interface Pointer, hresult	
CoCreateInstance.ind	FSPM	Host address, Class ID, Interface ID,	
CoCreateInstance.cnf(+)	FSPM	hresult, Interface Pointer	
CoCreateInstance.cnf(-)	FSPM	hresult	
CoDisconnectObject.ind	FSPM	Interface Pointer	
CoDisconnectObject.cnf	FSPM	Interface Pointer, hresult	
Call.ind	FSPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	
Call.cnf(+)	FSPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	
Call.cnf(-)	FSPM	Interface Pointer, Service Name, hresult	

5.5.2.2 Parameters of FAL User/FSPM Primitives

The parameters used with the primitives exchanged between the FAL User and the FSPM are described in FAL Service Definition in IEC 61158-5-10.

5.5.2.3 FSPM states

The defined state of the FSPM together with the description is listed in Table 213.

Table 213 – FSPM state descriptions

State Name	Description
ACTIVE	The FSPM in the ACTIVE state is ready to transmit and receive service primitives to and from the ARPM and the FAL user.

The state transition diagram of the FSPM is shown in Figure 45.

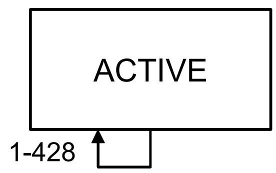

Figure 45 – State transition diagram of FSPM

5.5.2.4 FSPM state table

The FSPM state transitions are specified in Table 214.

Table 214 – FSPM state table

#	Current State	Event /Condition =>Action	Next State
1	ACTIVE	**QueryInterface.req(Interface Pointer, Interface ID)** => Interface Pointer:= Interface Pointer Service Name:=QueryInterface List of Unified Service In-Parameter:=Interface ID ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
2	ACTIVE	**QueryInterface.rsp(+)(Interface Pointer, hresult, ppvObject)** => Interface Pointer:= Interface Pointer Service Name:=QueryInterface hresult:=hresult List of Unified Service Out-Parameter:=ppvObject ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
3	ACTIVE	**QueryInterface.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=QueryInterface hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
4	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=QueryInterface => Interface Pointer:=Interface Pointer Interface ID:=List of Unified Service In-Parameter QueryInterface.ind(Interface Pointer, Interface ID)	ACTIVE
5	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=QueryInterface && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppvObject:=List of Unified Service Out-Parameter QueryInterface.cnf(+)(Interface Pointer, hresult, ppvObject)	ACTIVE
6	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=QueryInterface && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult QueryInterface.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
7	ACTIVE	**AddRef.req(Interface Pointer)** => Interface Pointer:= Interface Pointer Service Name:=AddRef List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
8	ACTIVE	**AddRef.rsp(Interface Pointer, Reference Count)** => Interface Pointer:= Interface Pointer Service Name:=AddRef hresult:="empty" List of Unified Service Out-Parameter:=Reference Count ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
9	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=AddRef => Interface Pointer:=Interface Pointer AddRef.ind(Interface Pointer)	ACTIVE
10	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AddRef => Interface Pointer:=Interface Pointer Reference Count:=List of Unified Service Out-Parameter AddRef.cnf(Interface Pointer, Reference Count)	ACTIVE
11	ACTIVE	**Release.req(Interface Pointer)** => Interface Pointer:= Interface Pointer Service Name:=Release List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
12	ACTIVE	**Release.rsp(Interface Pointer, Reference Count)** => Interface Pointer:= Interface Pointer Service Name:=Release hresult:="empty" List of Unified Service Out-Parameter:=Reference Count ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
13	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Release => Interface Pointer:=Interface Pointer Release.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
14	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Release => Interface Pointer:=Interface Pointer Reference Count:=List of Unified Service Out-Parameter Release.cnf(Interface Pointer, Reference Count)	ACTIVE
15	ACTIVE	**GetTypeInfoCount.req(Interface Pointer)** => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfoCount List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
16	ACTIVE	**GetTypeInfoCount.rsp(+)(Interface Pointer, hresult, pcTInfo)** => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfoCount hresult:=hresult List of Unified Service Out-Parameter:=pcTInfo ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
17	ACTIVE	**GetTypeInfoCount.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfoCount hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
18	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetTypeInfoCount => Interface Pointer:=Interface Pointer GetTypeInfoCount.ind(Interface Pointer)	ACTIVE
19	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetTypeInfoCount && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pcTInfo:=List of Unified Service Out-Parameter GetTypeInfoCount.cnf(+)(Interface Pointer, hresult, pcTInfo)	ACTIVE
20	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetTypeInfoCount && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetTypeInfoCount.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
21	ACTIVE	**GetTypeInfo.req(Interface Pointer, iTInfo, lcid)** => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfo List of Unified Service In-Parameter:=(iTInfo, lcid) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
22	ACTIVE	**GetTypeInfo.rsp(+)(Interface Pointer, hresult, ppTInfo)** => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfo hresult:=hresult List of Unified Service Out-Parameter:=ppTInfo ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
23	ACTIVE	**GetTypeInfo.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=GetTypeInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
24	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetTypeInfo => Interface Pointer:=Interface Pointer lcid:=List of Unified Service In-Parameter(lcid) iTInfo:=List of Unified Service In-Parameter(iTInfo) GetTypeInfo.ind(Interface Pointer, iTInfo, lcid)	ACTIVE
25	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetTypeInfo && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppTInfo:=List of Unified Service Out-Parameter GetTypeInfo.cnf(+)(Interface Pointer, hresult, ppTInfo)	ACTIVE
26	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetTypeInfo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetTypeInfo.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
27	ACTIVE	**GetIDsOfNames.req(Interface Pointer, riid, List of rgszNames, cNames, lcid)** => Interface Pointer:= Interface Pointer Service Name:=GetIDsOfNames List of Unified Service In-Parameter:=(riid, List of rgszNames, cNames, lcid) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
28	ACTIVE	**GetIDsOfNames.rsp(+)(Interface Pointer, hresult, List of rgDispId)** => Interface Pointer:= Interface Pointer Service Name:=GetIDsOfNames hresult:=hresult List of Unified Service Out-Parameter:=List of rgDispId ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
29	ACTIVE	**GetIDsOfNames.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=GetIDsOfNames hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
30	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetIDsOfNames => Interface Pointer:=Interface Pointer riid:=List of Unified Service In-Parameter(riid) List of rgszNames:=List of Unified Service In-Parameter(List of rgszNames) cNames:=List of Unified Service In-Parameter(cNames) lcid:=List of Unified Service In-Parameter(lcid) GetIDsOfNames.ind(Interface Pointer, riid, List of rgszNames, cNames, lcid)	ACTIVE
31	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetIDsOfNames && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of rgDispId:=List of Unified Service Out-Parameter GetIDsOfNames.cnf(+)(Interface Pointer, hresult, List of rgDispId)	ACTIVE
32	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetIDsOfNames && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetIDsOfNames.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
33	ACTIVE	**Invoke.req(Interface Pointer, dispIdMember, riid, lcid, wFlags, pDispParams)** => Interface Pointer:= Interface Pointer Service Name:=Invoke List of Unified Service In-Parameter:=(dispIdMember, riid, lcid, wFlags, pDispParams) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
34	ACTIVE	**Invoke.rsp(+)(Interface Pointer, hresult, pVarResult)** => Interface Pointer:= Interface Pointer Service Name:=Invoke hresult:=hresult List of Unified Service Out-Parameter:=pVarResult ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
35	ACTIVE	**Invoke.rsp(-)(Interface Pointer, hresult, pExcepinfo, puArgErr)** => Interface Pointer:= Interface Pointer Service Name:=Invoke hresult:=hresult List of Unified Service Out-Parameter:=(pExcepInfo, puArgErr) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
36	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Invoke => Interface Pointer:=Interface Pointer dispIdMember:=List of Unified Service In-Parameter(dispIdMember) riid:=List of Unified Service In-Parameter(riid) lcid:=List of Unified Service In-Parameter(lcid) wFlags:=List of Unified Service In-Parameter(wFlags) pDispParams:=List of Unified Service In-Parameter(pDispParams) Invoke.ind(Interface Pointer, dispIdMember, riid, lcid, wFlags, pDispParams)	ACTIVE
37	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Invoke && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVarResult:=List of Unified Service Out-Parameter(pVarResult) Invoke.cnf(+)(Interface Pointer, hresult, pVarResult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
38	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Invoke && hresult.S=Error && hresult<>DISP_E_EXCEPTION && hresult<>DISP_E_TYPEMISMATCH && hresult<>DISP_E_PARAMNOTFOUND => Interface Pointer:=Interface Pointer hresult:=hresult pExcepInfo:="empty" puArgErr:="empty" Invoke.cnf(-)(Interface Pointer, hresult, pExcepInfo, puArgErr)	ACTIVE
39	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Invoke && (hresult=DISP_E_TYPEMISMATCH || hresult=DISP_E_PARAMNOTFOUND) => Interface Pointer:=Interface Pointer hresult:=hresult pExcepInfo:="empty" puArgErr:=List of Unified Service Out-Parameter(puArgErr) Invoke.cnf(-)(Interface Pointer, hresult, pExcepInfo, puArgErr)	ACTIVE
40	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Invoke && hresult=DISP_E_EXCEPTION => Interface Pointer:=Interface Pointer hresult:=hresult pExcepInfo:=List of Unified Service Out-Parameter(pExcepInfo) puArgErr:="empty" Invoke.cnf(-)(Interface Pointer, hresult, pExcepInfo, puArgErr)	ACTIVE
41	ACTIVE	**get_Producer.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_Producer List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
42	ACTIVE	**get_Producer.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:= Interface Pointer Service Name:=get_Producer hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
43	ACTIVE	**get_Producer.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=get_Producer hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
44	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_Producer => Interface Pointer:=Interface Pointer get_Producer.ind(Interface Pointer)	ACTIVE
45	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Producer && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Producer.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
46	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Producer && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Producer.cnf(-)(Interface Pointer, hresult)	ACTIVE
47	ACTIVE	**get_Product.req(Interface Pointer)** => Interface Pointer:= Interface Pointer Service Name:=get_Product List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
48	ACTIVE	**get_Product.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:= Interface Pointer Service Name:=get_Product hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
49	ACTIVE	**get_Product.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=get_Product hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
50	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_Product => Interface Pointer:=Interface Pointer get_Product.ind(Interface Pointer)	ACTIVE
51	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Product&& hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Product.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
52	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Product && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Product.cnf(-)(Interface Pointer, hresult)	ACTIVE
53	ACTIVE	**get_SerialNo.req(Interface Pointer)** => Interface Pointer:= Interface Pointer Service Name:=get_SerialNo List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
54	ACTIVE	**get_SerialNo.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:= Interface Pointer Service Name:=get_SerialNo hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
55	ACTIVE	**get_SerialNo.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=get_SerialNo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
56	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_SerialNo => Interface Pointer:=Interface Pointer get_SerialNo.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
57	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_SerialNo&& hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_SerialNo.cnf(+)(Interface Pointer,hresult, pVal)	ACTIVE
58	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_SerialNo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_SerialNo.cnf(-)(Interface Pointer, hresult)	ACTIVE
59	ACTIVE	**get_ProductionDate.req(Interface Pointer)** => Interface Pointer:= Interface Pointer Service Name:=get_Product List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
60	ACTIVE	**get_ProductionDate.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_ProductionDate hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
61	ACTIVE	**get_ProductionDate.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_ProductionDate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
62	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_ProductionDate => Interface Pointer:=Interface Pointer get_ProductionDate.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
63	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_ProductionDate && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_ProductionDate.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
64	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_ProductionDate && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_ProductionDate.cnf(-)(Interface Pointer, hresult)	ACTIVE
65	ACTIVE	**Revision.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Revision List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
66	ACTIVE	**Revision.rsp(+)(Interface Pointer, hresult, pMajor, pMinor)** => Interface Pointer:=Interface Pointer Service Name:=Revision hresult:=hresult List of Unified Service Out-Parameter:=(pMajor, pMinor) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
67	ACTIVE	**Revision.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Revision hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
68	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Revision => Interface Pointer:=Interface Pointer Revision.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
69	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Revision && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMajor:=List of Unified Service Out-Parameter(pMajor) pMinor:=List of Unified Service Out-Parameter(pMinor) Revision.cnf(+)(Interface Pointer, hresult, pMajor, pMinor)	ACTIVE
70	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Revision && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Revision.cnf(-)(Interface Pointer, hresult)	ACTIVE
71	ACTIVE	**get_LogicalDevice.req(Interface Pointer, Name)** => Interface Pointer:= Interface Pointer Service Name:=get_LogicalDevice List of Unified Service In-Parameter:=(Name) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
72	ACTIVE	**get_LogicalDevice.rsp(+)(Interface Pointer, hresult, ppLDev)** => Interface Pointer:=Interface Pointer Service Name:=get_LogicalDevice hresult:=hresult List of Unified Service Out-Parameter:=ppLDev ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
73	ACTIVE	**get_LogicalDevice.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_LogicalDevice hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
74	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_LogicalDevice => Interface Pointer:=Interface Pointer Name:=List of Unified Service In-Parameter get_LogicalDevice.ind(Interface Pointer, Name)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
75	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_LogicalDevice && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppLDev:=List of Unified Service Out-Parameter get_LogicalDevice.cnf(+)(Interface Pointer, hresult, ppLDev)	ACTIVE
76	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_LogicalDevice && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_LogicalDevice.cnf(-)(Interface Pointer, hresult)	ACTIVE
77	ACTIVE	**Type.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Type List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
78	ACTIVE	**Type.rsp(+)(Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack)** => Interface Pointer:=Interface Pointer Service Name:=Type hresult:=hresult List of Unified Service Out-Parameter:=(pMultiApp, pPROFInetDcomStack) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
79	ACTIVE	**Type.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Type hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
80	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Type => Interface Pointer:=Interface Pointer Type.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
81	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Type && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMultiApp:=List of Unified Service Out-Parameter(pMultiApp) pPROFInetDcomStack:=List of Unified Service Out-Parameter(pPROFInetDcomStack) Type.cnf(+)(Interface Pointer, hresult, pMultiApp, pPROFInetDcomStack)	ACTIVE
82	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Type && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Type.cnf(-)(Interface Pointer, hresult)	ACTIVE
83	ACTIVE	**PROFInetRevision.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=PROFInetRevision List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
84	ACTIVE	**PROFInetRevision.rsp(+)(Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild)** => Interface Pointer:=Interface Pointer Service Name:=PROFInetRevision hresult:=hresult List of Unified Service Out-Parameter:=(pMajor, pMinor, pServicePack, pBuild) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
85	ACTIVE	**PROFInetRevision.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=PROFInetRevision hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
86	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=PROFInetRevision => Interface Pointer:=Interface Pointer PROFInetRevision.cnf(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
87	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=PROFInetRevision && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMajor:=List of Unified Service Out-Parameter(pMajor) pMinor:=List of Unified Service Out-Parameter(pMinor) pServicePack:=List of Unified Service Out-Parameter(pServicePack) pBuild:=List of Unified Service Out-Parameter(pBuild) PROFInetRevision.cnf(+)(Interface Pointer, hresult, pMajor, pMinor, pServicePack, pBuild)	ACTIVE
88	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=PROFInetRevision && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult PROFInetRevision.cnf(-)(Interface Pointer, hresult)	ACTIVE
89	ACTIVE	**get_PDevStamp.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_PDevStamp List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
90	ACTIVE	**get_PDevStamp.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_PDevStamp hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
91	ACTIVE	**get_PDevStamp.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_PDevStamp hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
92	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_PDevStamp => Interface Pointer:=Interface Pointer get_PDevStamp.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
93	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_PDevStamp && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_PDevStamp.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
94	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_PDevStamp && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_PDevStamp.cnf(-)(Interface Pointer, hresult)	ACTIVE
95	ACTIVE	**get_Count.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_Count List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
96	ACTIVE	**get_Count.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_Count hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
97	ACTIVE	**get_Count.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_Count hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
98	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_Count => Interface Pointer:=Interface Pointer get_Count.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
99	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Count && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Count.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
100	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Count && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Count.cnf(-)(Interface Pointer, hresult)	ACTIVE
101	ACTIVE	**BrowseItems.req(Interface Pointer, Offset, MaxReturn)** => Interface Pointer:=Interface Pointer Service Name:=BrowseItems List of Unified Service In-Parameter:=(Offset, MaxReturn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
102	ACTIVE	**BrowseItems.rsp(+)(Interface Pointer, hresult, pItem, pDataType, pAccessRight)** => Interface Pointer:=Interface Pointer Service Name:=BrowseItem hresult:=hresult List of Unified Service Out-Parameter:=(pItem, pDataType, pAccessRight) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
103	ACTIVE	**BrowseItems.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=BrowseItems hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
104	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=BrowseItems => Interface Pointer:=Interface Pointer Offset:=List of Unified Service In-Parameter(Offset) MaxReturn:=List of Unified Service In-Parameter(MaxReturn) BrowseItem.ind(Interface Pointer, Offset, MaxReturn)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
105	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=BrowseItems && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pItem:=List of Unified Service Out-Parameter(pItem) pDataType:=List of Unified Service Out-Parameter(pDataType) pAccessRights:=List of Unified Service Out-Parameter(pAccessRights) BrowseItem.cnf(+)(Interface Pointer, hresult, pItem, pDataType, pAccessRight)	ACTIVE
106	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=BrowseItems && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult BrowseItem.cnf(-)(Interface Pointer, hresult)	ACTIVE
107	ACTIVE	**get_Count2.req(Interface Pointer, Selector)** => Interface Pointer:=Interface Pointer Service Name:=get_Count2 List of Unified Service In-Parameter:=Selector ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
108	ACTIVE	**get_Count2.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_Count2 hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
109	ACTIVE	**get_Count2.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_Count2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
110	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_Count2 => Interface Pointer:=Interface Pointer Selector:=List of Unified Service In-Parameter get_Count2.ind(Interface Pointer, Selector)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
111	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Count2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Count2.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
112	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Count2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Count2.cnf(-)(Interface Pointer, hresult)	ACTIVE
113	ACTIVE	**BrowseItems2.req(Interface Pointer, Selector, Offset, MaxReturn)** => Interface Pointer:=Interface Pointer Service Name:=BrowseItems2 List of Unified Service In-Parameter:=(Selector, Offset, MaxReturn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
114	ACTIVE	**BrowseItems2.rsp(+)(Interface Pointer, hresult, pItem, pInfo1, pInfo2)** => Interface Pointer:=Interface Pointer Service Name:=BrowseItem2 hresult:=hresult List of Unified Service Out-Parameter:=(pItem, pInfo1, pInfo2) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
115	ACTIVE	**BrowseItems2.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=BrowseItems2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
116	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=BrowseItems2 => Interface Pointer:=Interface Pointer Selector:=List of Unified Service In-Parameter(Selector) Offset:=List of Unified Service In-Parameter(Offset) MaxReturn:=List of Unified Service In-Parameter(MaxReturn) BrowseItems2.ind(Interface Pointer, Selector, Offset, MaxReturn)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
117	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=BrowseItems2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pItem:=List of Unified Service Out-Parameter(pItem) pInfo1:=List of Unified Service Out-Parameter(pInfo1) pInfo2:=List of Unified Service Out-Parameter(pInfo2) BrowseItems2.cnf(+)(Interface Pointer, hresult, pItem, pInfo1, pInfo2)	ACTIVE
118	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=BrowseItems2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult BrowseItems2.cnf(-)(Interface Pointer, hresult)	ACTIVE
119	ACTIVE	**Save.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Save List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
120	ACTIVE	**Save.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Save hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
121	ACTIVE	**Save.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Save hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
122	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Save => Interface Pointer:=Interface Pointer Save.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
123	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Save && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Save.cnf(+)(Interface Pointer, hresult)	ACTIVE
124	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Save && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Save.cnf(-)(Interface Pointer, hresult)	ACTIVE
125	ACTIVE	**Save2.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Save2 List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
126	ACTIVE	**Save2.rsp(+)(Interface Pointer, hresult, pLDevName, pPartialResult)** => Interface Pointer:=Interface Pointer Service Name:=Save2 hresult:=hresult List of Unified Service Out-Parameter:=(pLDevName, pPartialResult) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
127	ACTIVE	**Save2.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Save2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
128	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Save2 => Interface Pointer:=Interface Pointer Save2.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
129	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Save2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pLDevName:=List of Unified Service Out-Parameter(pLDevName) pPartialResult:=List of Unified Service Out-Parameter(pPartialResult) Save2.cnf(+)(Interface Pointer, hresult, pLDevName, pPartialResult)	ACTIVE
130	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Save2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Save2.cnf(-)(Interface Pointer, hresult)	ACTIVE
131	ACTIVE	**get_Name.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_Name List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
132	ACTIVE	**get_Name.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_Name hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
133	ACTIVE	**get_Name.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_Name hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
134	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_Name => Interface Pointer:=Interface Pointer get_Name.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
135	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Name && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Name.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
136	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Name && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Name.cnf(-)(Interface Pointer, hresult)	ACTIVE
137	ACTIVE	**get_ACCO.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_ACCO List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
138	ACTIVE	**get_ACCO.rsp(+)(Interface Pointer, hresult, ppACCO)** => Interface Pointer:=Interface Pointer Service Name:=get_ACCO hresult:=hresult List of Unified Service Out-Parameter:=ppACCO ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
139	ACTIVE	**get_ACCO.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_ACCO hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
140	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_ACCO => Interface Pointer:=Interface Pointer get_ACCO.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
141	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_ACCO && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppACCO:=List of Unified Service Out-Parameter get_ACCO.cnf(+)(Interface Pointer, hresult, ppACCO)	ACTIVE
142	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_ACCO && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_ACCO.cnf(-)(Interface Pointer, hresult)	ACTIVE
143	ACTIVE	**get_RTAuto.req(Interface Pointer, Name)** => Interface Pointer:=Interface Pointer Service Name:=get_RTAuto List of Unified Service In-Parameter:=Name ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
144	ACTIVE	**get_RTAuto.rsp(+)(Interface Pointer, hresult, ppAuto)** => Interface Pointer:=Interface Pointer Service Name:=get_RTAuto hresult:=hresult List of Unified Service Out-Parameter:=ppAuto ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
145	ACTIVE	**get_RTAuto.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_RTAuto hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
146	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_RTAuto => Interface Pointer:=Interface Pointer Name:=List of Unified Service InParameter get_RTAuto.ind(Interface Pointer, Name)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
147	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_RTAuto && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppAuto:=List of Unified Service Out-Parameter get_RTAuto.cnf(+)(Interface Pointer, hresult, ppAuto)	ACTIVE
148	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_RTAuto && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_RTAuto.cnf(-)(Interface Pointer, hresult)	ACTIVE
149	ACTIVE	**ComponentInfo.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=ComponentInfo List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
150	ACTIVE	**ComponentInfo.rsp(+)(Interface Pointer, hresult, pComponentID, pVersion)** => Interface Pointer:=Interface Pointer Service Name:=ComponentInfo hresult:=hresult List of Unified Service Out-Parameter:=(pComponentID, pVersion) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
151	ACTIVE	**ComponentInfo.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=ComponentInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
152	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=ComponentInfo => Interface Pointer:=Interface Pointer ComponentInfo.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
153	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ComponentInfo && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pComponentID:=List of Unified Service Out-Parameter(pComponentID) pVersion:=List of Unified Service Out-Parameter(pVersion) ComponentInfo.cnf(+)(Interface Pointer, hresult, pComponentID, pVersion)	ACTIVE
154	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ComponentInfo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ComponentInfo.cnf(-)(Interface Pointer, hresult)	ACTIVE
155	ACTIVE	**get_State.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_State List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
156	ACTIVE	**get_State.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_State hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
157	ACTIVE	**get_State.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_State hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
158	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_State => Interface Pointer:=Interface Pointer get_State.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
159	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_State && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_State.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
160	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_State && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_State.cnf(-)(Interface Pointer, hresult)	ACTIVE
161	ACTIVE	**Activate.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Activate List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
162	ACTIVE	**Activate.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Activate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
163	ACTIVE	**Activate.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Activate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
164	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Activate => Interface Pointer:=Interface Pointer Activate.ind(Interface Pointer)	ACTIVE
165	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Activate && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Activate.cnf(+)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
166	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Activate && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Activate.cnf(-)(Interface Pointer, hresult)	ACTIVE
167	ACTIVE	**Deactivate.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Deactivate List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
168	ACTIVE	**Deactivate.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Deactivate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
169	ACTIVE	**Deactivate.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Deactivate hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
170	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Deactivate => Interface Pointer:=Interface Pointer Deactivate.ind(Interface Pointer)	ACTIVE
171	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Deactivate && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Deactivate.cnf(+)(Interface Pointer, hresult)	ACTIVE
172	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Deactivate && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Deactivate.cnf(-)(Interface Pointer, hresult)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
173	ACTIVE	**Reset.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Reset List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
174	ACTIVE	**Reset.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Reset hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
175	ACTIVE	**Reset.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Reset hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
176	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Reset => Interface Pointer:=Interface Pointer Reset.ind(Interface Pointer)	ACTIVE
177	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Reset && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Reset.cnf(+)(Interface Pointer, hresult)	ACTIVE
178	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Reset && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Reset.cnf(-)(Interface Pointer, hresult)	ACTIVE
179	ACTIVE	**AdviseState.req(Interface Pointer, pStateEvent)** => Interface Pointer:=Interface Pointer Service Name:=AdviseState List of Unified Service In-Parameter:=pStateEvent ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
180	ACTIVE	**AdviseState.rsp(+)(Interface Pointer, hresult, pCookie)** => Interface Pointer:=Interface Pointer Service Name:=AdviseState hresult:=hresult List of Unified Service Out-Parameter:=pCookie ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
181	ACTIVE	**AdviseState.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=AdviseState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
182	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=AdviseState => Interface Pointer:=Interface Pointer pStateEvent:=List of Unified Service In-Parameter AdviseState.ind(Interface Pointer, pStateEvent)	ACTIVE
183	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AdviseState && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCookie:=List of Unified Service Out-Parameter AdviseState.cnf(+)(Interface Pointer, hresult, pCookie)	ACTIVE
184	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AdviseState && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult AdviseState.cnf(-)(Interface Pointer, hresult)	ACTIVE
185	ACTIVE	**UnadviseState.req(Interface Pointer, Cookie)** => Interface Pointer:=Interface Pointer Service Name:=UnadviseState List of Unified Service In-Parameter:=Cookie ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
186	ACTIVE	**UnadviseState.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=UnadviseState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
187	ACTIVE	**UnadviseState.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=UnadviseState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
188	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=UnadviseState => Interface Pointer:=Interface Pointer Cookie:=List of Unified Service In-Parameter UnadviseState.ind(Interface Pointer, Cookie)	ACTIVE
189	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=UnadviseState && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseState.cnf(+)(Interface Pointer, hresult)	ACTIVE
190	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=UnadviseState && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseState.cnf(-)(Interface Pointer, hresult)	ACTIVE
191	ACTIVE	**get_Time.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_Time List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
192	ACTIVE	**get_Time.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_Time hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
193	ACTIVE	**get_Time.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_Time hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
194	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_Time => Interface Pointer:=Interface Pointer get_Time.ind(Interface Pointer)	ACTIVE
195	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Time && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_Time.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
196	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_Time && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_Time.cnf(-)(Interface Pointer, hresult)	ACTIVE
197	ACTIVE	**put_Time.req(Interface Pointer, NewVal)** => Interface Pointer:=Interface Pointer Service Name:=put_Time List of Unified Service In-Parameter:=NewVal ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
198	ACTIVE	**put_Time.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=put_Time hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
199	ACTIVE	**put_Time.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=put_Time hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
200	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=put_Time => Interface Pointer:=Interface Pointer NewVal:=List of Unified Service In-Parameter put_Time.ind(Interface Pointer, NewVal)	ACTIVE
201	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=put_Time && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult put_Time.cnf(+)(Interface Pointer, hresult)	ACTIVE
202	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=put_Time && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult put_Time.cnf(-)(Interface Pointer, hresult)	ACTIVE
203	ACTIVE	**GroupError.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=GroupError List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
204	ACTIVE	**GroupError.rsp(+)(Interface Pointer, hresult, pVal, pMagicCookie)** => Interface Pointer:=Interface Pointer Service Name:=GroupError hresult:=hresult List of Unified Service Out-Parameter:=(pVal, pMagicCookie) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
205	ACTIVE	**GroupError.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
206	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GroupError => Interface Pointer:=Interface Pointer GroupError.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
207	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GroupError && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter(pVal) pMagicCookie:=List of Unified Service Out-Parameter(pMagicCookie) GroupError.cnf(+)(Interface Pointer, hresult, pVal, pMagicCookie)	ACTIVE
208	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GroupError && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GroupError.cnf(-)(Interface Pointer, hresult)	ACTIVE
209	ACTIVE	**AdviseGroupError.req(Interface Pointer, pGroupErrorEvent)** => Interface Pointer:=Interface Pointer Service Name:=AdviseGroupError List of Unified Service In-Parameter:=pGroupErrorEvent ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
210	ACTIVE	**AdviseGroupError.rsp(+)(Interface Pointer, hresult, pCookie)** => Interface Pointer:=Interface Pointer Service Name:=AdviseGroupError hresult:=hresult List of Unified Service Out-Parameter:=pCookie ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
211	ACTIVE	**AdviseGroupError.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=AdviseGroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
212	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=AdviseGroupError => Interface Pointer:=Interface Pointer pGroupErrorEvent:=List of Unified Service In-Parameter AdviseGroupError.ind(Interface Pointer, pGroupErrorEvent)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
213	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AdviseGroupError && hresult.S=Success || S_FALSE => Interface Pointer:=Interface Pointer hresult:=hresult pCookie:=List of Unified Service Out-Parameter AdviseGroupError.cnf(+)(Interface Pointer, hresult, pCookie)	ACTIVE
214	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AdviseGroupError && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult AdviseGroupError.cnf(-)(Interface Pointer, hresult)	ACTIVE
215	ACTIVE	**UnadviseGroupError.req(Interface Pointer, Cookie)** => Interface Pointer:=Interface Pointer Service Name:=UnadviseGroupError List of Unified Service In-Parameter:=Cookie ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
216	ACTIVE	**UnadviseGroupError.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=UnadviseGroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
217	ACTIVE	**UnadviseGroupError.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=UnadviseGroupError hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
218	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=UnadviseGroupError => Interface Pointer:=Interface Pointer Cookie:=List of Unified Service In-Parameter UnadviseGroupError.ind(Interface Pointer, Cookie)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
219	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=UnadviseGroupError && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseGroupError.cnf(+)(Interface Pointer, hresult)	ACTIVE
220	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=UnadviseGroupError && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult UnadviseGroupError.cnf(-)(Interface Pointer, hresult)	ACTIVE
221	ACTIVE	**AddConnections.req(Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn)** => Interface Pointer:=Interface Pointer Service Name:=AddConnections List of Unified Service In-Parameter:=(Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
222	ACTIVE	**AddConnections.rsp(+)(Interface Pointer, hresult, List of ppAddConnectionOut)** => Interface Pointer:=Interface Pointer Service Name:=AddConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppAddConnectionOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
223	ACTIVE	**AddConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=AddConnection hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
224	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=AddConnections => Interface Pointer:=Interface Pointer Provider:=List of Unified Service In-Parameter(Provider) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) State:=List of Unified Service In-Parameter(State) Count:=List of Unified Service In-Parameter(Count) List of pAddConnectionIn:=List of Unified Service In-Parameter(List of pAddConnectionIn) AddConnections.ind(Interface Pointer, Provider, QoSType, QoSValue, State, Count, List of pAddConnectionIn)	ACTIVE
225	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AddConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ppAddConnectionOut:=List of Unified Service Out-Parameter AddConnections.cnf(+)(Interface Pointer, hresult, List of ppAddConnectionOut)	ACTIVE
226	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AddConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult AddConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
227	ACTIVE	**RemoveConnections.req(Interface Pointer, Count, List of pConsumerID)** => Interface Pointer:=Interface Pointer Service Name:=RemoveConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
228	ACTIVE	**RemoveConnections.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=RemoveConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
229	ACTIVE	**RemoveConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=RemoveConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
230	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=RemoveConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameterer(List of pConsumerID) RemoveConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE
231	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=RemoveConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter RemoveConnections.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
232	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=RemoveConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult RemoveConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
233	ACTIVE	**ClearConnections.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=ClearConnections List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
234	ACTIVE	**ClearConnections.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=ClearConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
235	ACTIVE	**ClearConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=ClearConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
236	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=ClearConnections => Interface Pointer:=Interface Pointer ClearConnections.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
237	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ClearConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult ClearConnections.cnf(+)(Interface Pointer, hresult)	ACTIVE
238	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ClearConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ClearConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
239	ACTIVE	**SetActivationState.req(Interface Pointer, State, Count, List of pConsumerID)** => Interface Pointer:=Interface Pointer Service Name:=SetActivationState List of Unified Service In-Parameter:=(State, Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
240	ACTIVE	**SetActivationState.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=SetActivationState hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
241	ACTIVE	**SetActivationState.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=SetActivationState hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
242	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=SetActivationState => Interface Pointer:=Interface Pointer State:=List of Unified Service In-Parameter(State) Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) SetActivationState.ind(Interface Pointer, State, Count, List of pConsumerID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
243	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=SetActivationState && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter SetActivationState.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
244	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=AddConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult SetActivationState.cnf(-)(Interface Pointer, hresult)	ACTIVE
245	ACTIVE	**GetInfo.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=GetInfo List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
246	ACTIVE	**GetInfo.rsp(+)(Interface Pointer, hresult, pMax, pCurCnt)** => Interface Pointer:=Interface Pointer Service Name:=GetInfo hresult:=hresult List of Unified Service Out-Parameter:=(pMax, pCurCnt) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
247	ACTIVE	**GetInfo.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
248	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetInfo => Interface Pointer:=Interface Pointer GetInfo.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
249	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetInfo && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pMax:=List of Unified Service Out-Parameter(pMax) pCurCnt:=List of Unified Service Out-Parameter(pCurCnt) GetInfo.cnf(+)(Interface Pointer, hresult, pMax, pCurCnt)	ACTIVE
250	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetInfo && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetInfo.cnf(-)(Interface Pointer, hresult)	ACTIVE
251	ACTIVE	**GetIDs.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=GetIDs List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
252	ACTIVE	**GetIDs.rsp(+)(Interface Pointer, hresult, pCount, List of ppGetIDOut)** => Interface Pointer:=Interface Pointer Service Name:=GetIDs hresult:=hresult List of Unified Service Out-Parameter:=(pCount, List of ppGetIDOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
253	ACTIVE	**GetIDs.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetInfo hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
254	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetIDs => Interface Pointer:=Interface Pointer GetIDs.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
255	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetIDs && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCount:=List of Unified Service Out-Parameter(pCount) List of ppGetIDOut:=List of Unified Service Out-Parameter(List of ppGetIDOut) GetIDs.cnf(+)(Interface Pointer, hresult, pCount, List of ppGetIDOut)	ACTIVE
256	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetIDs && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetIDs.cnf(-)(Interface Pointer, hresult)	ACTIVE
257	ACTIVE	**GetConnections.req(Interface Pointer, Count, List of pConsumerID)** => Interface Pointer:=Interface Pointer Service Name:=GetConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
258	ACTIVE	**GetConnections.rsp(+)(Interface Pointer, hresult, List of ppGetConnectionOut)** => Interface Pointer:=Interface Pointer Service Name:=GetConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppGetConnectionOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
259	ACTIVE	**GetConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
260	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) GetConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
261	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppGetConnectionsOut:=List of Unified Service Out-Parameter GetConnections.cnf(+)(Interface Pointer, hresult, List of ppGetConnectionsOut)	ACTIVE
262	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
263	ACTIVE	**ReviseQoS.req(Interface Pointer, RTAuto, QoSType, QoSValue)** => Interface Pointer:=Interface Pointer Service Name:=ReviseQoS List of Unified Service In-Parameter:=(RTAuto, QoSType, QoSValue) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
264	ACTIVE	**ReviseQoS.rsp(+)(Interface Pointer, hresult, pRevisedQoSValue)** => Interface Pointer:=Interface Pointer Service Name:=ReviseQoS hresult:=hresult List of Unified Service Out-Parameter:=ppRevisedQoSValue ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
265	ACTIVE	**ReviseQoS.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=RevisedQoS hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
266	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=RevisedQoS => Interface Pointer:=Interface Pointer RTAuto:=List of Unified Service In-Parameter(RTAuto) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) ReviseQoS.ind(Interface Pointer, RTAuto, QoSType, QoSValue)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
267	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=RevisedQoS && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pRevisedQoSValue:=List of Unified Service Out-Parameter RevisedQoS.cnf(+)(Interface Pointer, hresult, pRevisedQoSValue)	ACTIVE
268	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=RevisedQoS && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult RevisedQoS.cnf(-)(Interface Pointer, hresult)	ACTIVE
269	ACTIVE	**get_PingFactor.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_PingFactor List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
270	ACTIVE	**get_PingFactor.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_PingFactor hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
271	ACTIVE	**get_PingFactor.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_PingFactor hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
272	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_PingFactor => Interface Pointer:=Interface Pointer get_PingFactor.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
273	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_PingFactor && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_PingFactor.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE
274	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_PingFactor && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_PingFactor.cnf(-)(Interface Pointer, hresult)	ACTIVE
275	ACTIVE	**put_PingFactor.req(Interface Pointer, NewVal)** => Interface Pointer:=Interface Pointer Service Name:=put_PingFactor List of Unified Service In-Parameter:=NewVal ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
276	ACTIVE	**put_PingFactor.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=put_PingFactor hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
277	ACTIVE	**put_PingFactor.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=put_PingFactor hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
278	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=put_PingFactor => Interface Pointer:=Interface Pointer NewVal:=List of Unified Service In-Parameter put_PingFactor.ind(Interface Pointer, NewVal)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
279	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=put_PingFactor && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult put_PingFactor.cnf(+)(Interface Pointer, hresult)	ACTIVE
280	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=put_PingFactor && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult put_PingFactor.cnf(-)(Interface Pointer, hresult)	ACTIVE
281	ACTIVE	**get_CDBCookie.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=get_CDBCookie List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
282	ACTIVE	**get_CDBCookie.rsp(+)(Interface Pointer, hresult, pVal)** => Interface Pointer:=Interface Pointer Service Name:=get_CDBCookie hresult:=hresult List of Unified Service Out-Parameter:=pVal ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
283	ACTIVE	**get_CDBCookie.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=get_CDBCookie hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
284	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=get_CDBCookie => Interface Pointer:=Interface Pointer get_CDBCookie.ind(Interface Pointer)	ACTIVE
285	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_CDBCookie && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pVal:=List of Unified Service Out-Parameter get_CDBCookie.cnf(+)(Interface Pointer, hresult, pVal)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
286	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=get_CDBCookie && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult get_CDBCookie.cnf(-)(Interface Pointer, hresult)	ACTIVE
287	ACTIVE	**GetConsIDs.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=GetConsIDs List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
288	ACTIVE	**GetConsIDs.rsp(+)(Interface Pointer, hresult, pCount, List of ppConsumerID)** => Interface Pointer:=Interface Pointer Service Name:=GetConsIDs hresult:=hresult List of Unified Service Out-Parameter:=(pCount, List of ppConsumerID) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
289	ACTIVE	**GetConsIDs.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetConsIDs hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
290	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetConsIDs => Interface Pointer:=Interface Pointer GetConsIDs.ind(Interface Pointer)	ACTIVE
291	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConsIDs && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCount:=List of Unified Service Out-Parameter(pCount) List of ppConsumerID:=List of Unified Service Out-Parameter(List of ppConsumerID) GetConsIDs.cnf(+)(Interface Pointer, hresult, pCount, List of ppConsumerID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
292	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConsIDs && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConsIDs.cnf(-)(Interface Pointer, hresult)	ACTIVE
293	ACTIVE	**GetConsConnections.req(Interface Pointer, Count, List of pConsumerID)** => Interface Pointer:=Interface Pointer Service Name:=GetConsConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
294	ACTIVE	**GetConsConnections.rsp(+)(Interface Pointer, hresult, List of ppGetConsConnOut)** => Interface Pointer:=Interface Pointer Service Name:=GetConsConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppGetConsConnOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
295	ACTIVE	**GetConsConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetConsConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
296	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetConsConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) GetConsConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE
297	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConsConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppGetConsConnOut:=List of Unified Service Out-Parameter GetConsConnections.cnf(+)(Interface Pointer, hresult, List of ppGetConsConnOut)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
298	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConsConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConsConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
299	ACTIVE	**DiagConsConnections.req(Interface Pointer, Count, List of pConsumerID)** => Interface Pointer:=Interface Pointer Service Name:=DiagConsConnections List of Unified Service In-Parameter:=(Count, List of pConsumerID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
300	ACTIVE	**DiagConsConnections.rsp(+)(Interface Pointer, hresult, List of ppDiagConsConnOut)** => Interface Pointer:=Interface Pointer Service Name:=DiagConsConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppDiagConsConnOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
301	ACTIVE	**DiagConsConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=DiagConsConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
302	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=DiagConsConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pConsumerID:=List of Unified Service In-Parameter(List of pConsumerID) DiagConsConnections.ind(Interface Pointer, Count, List of pConsumerID)	ACTIVE
303	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=DiagConsConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppDiagConsConnOut:=List of Unified Service Out-Parameter DiagConsConnections.cnf(+)(Interface Pointer, hresult, List of ppDiagConsConnOut)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
304	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=DiagConsConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult DiagConsConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
305	ACTIVE	**GetProvIDs.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=GetProvIDs List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
306	ACTIVE	**GetProvIDs.rsp(+)(Interface Pointer, hresult, pCount, List of ppProviderID)** => Interface Pointer:=Interface Pointer Service Name:=GetProvIDs hresult:=hresult List of Unified Service Out-Parameter:=(pCount, List of ppProviderID) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
307	ACTIVE	**GetProvIDs.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetProvIDs hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
308	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetProvIDs => Interface Pointer:=Interface Pointer GetProvIDs.ind(Interface Pointer)	ACTIVE
309	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetProvIDs && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pCount:=List of Unified Service Out-Parameter(pCount) List of ppProviderID:=List of Unified Service Out-Parameter(List of ppProviderID) GetProvIDs.cnf(+)(Interface Pointer, hresult, pCount, List of ppProviderID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
310	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetProvIDs && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetProvIDs.cnf(-)(Interface Pointer, hresult)	ACTIVE
311	ACTIVE	**GetProvConnections.req(Interface Pointer, Count, List of pProviderID)** => Interface Pointer:=Interface Pointer Service Name:=GetProvConnections List of Unified Service In-Parameter:=(Count, List of pProviderID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
312	ACTIVE	**GetProvConnections.rsp(+)(Interface Pointer, hresult, List of ppGetProvConnOut)** => Interface Pointer:=Interface Pointer Service Name:=GetProvConnections hresult:=hresult List of Unified Service Out-Parameter:=List of ppGetProvConnOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
313	ACTIVE	**GetProvConnections.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetProvConnections hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
314	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetProvConnections => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pProviderID:=List of Unified Service In-Parameter(List of pProviderID) GetProvConnections.ind(Interface Pointer, Count, List of pProviderID)	ACTIVE
315	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetProvConnections && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppDiagConsConnOut:=List of Unified Service Out-Parameter GetProvConnections.cnf(+)(Interface Pointer, hresult, List of ppGetProvConnOut)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
316	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetProvConnections && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetProvConnections.cnf(-)(Interface Pointer, hresult)	ACTIVE
317	ACTIVE	**GetDiagnosis.req(Interface Pointer, Request, InLength, pInBuffer)** => Interface Pointer:=Interface Pointer Service Name:=GetDiagnosis List of Unified Service In-Parameter:=(Request, InLength, pInBuffer) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
318	ACTIVE	**GetDiagnosis.rsp(+)(Interface Pointer, hresult, pOutLength, ppOutBuffer)** => Interface Pointer:=Interface Pointer Service Name:=GetDiagnosis hresult:=hresult List of Unified Service Out-Parameter:=(pOutLength, ppOutBuffer) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
319	ACTIVE	**GetDiagnosis.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetDiagnosis hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
320	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetDiagnosis => Interface Pointer:=Interface Pointer Request:=List of Unified Service In-Parameter(Request) InLength:=List of Unified Service In-Parameter(InLength) pInBuffer:=List of Unified Service In-Parameter(pInBuffer) GetDiagnosis.ind(Interface Pointer, Request, InLength, pInBuffer)	ACTIVE
321	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetDiagnosis && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pOutLength:=List of Unified Service Out-Parameter(pOutLength) ppOutBuffer:=List of Unified Service Out-Parameter(ppOutBuffer) GetDiagnosis.cnf(+)(Interface Pointer, hresult, pOutLength, ppOutBuffer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
322	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetDiagnosis && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetDiagnosis.cnf(-)(Interface Pointer, hresult)	ACTIVE
323	ACTIVE	**Connect.req(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn)** => Interface Pointer:=Interface Pointer Service Name:=Connect List of Unified Service In-Parameter:=(Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
324	ACTIVE	**Connect.rsp(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut)** => Interface Pointer:=Interface Pointer Service Name:=Connect hresult:=hresult List of Unified Service Out-Parameter:=(pFirstConnect, List of ppConnectOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
325	ACTIVE	**Connect.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Connect hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
326	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Connect => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter(Consumer) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) State:=List of Unified Service In-Parameter(State) pICBACallback:=List of Unified Service In-Parameter(pICBACallback) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) Connect.ind(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallBack, Count, List of pConnectIn)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
327	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Connect && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pFirstConnect:=List of Unified Service Out-Parameter(pFirstConnect) List of ppConnectOut:=List of Unified Service Out-Parameter(List of ppConnectOut) Connect.cnf(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut)	ACTIVE
328	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Connect && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Connect.cnf(-)(Interface Pointer, hresult)	ACTIVE
329	ACTIVE	**Disconnect.req(Interface Pointer, Count, List of pProviderID)** => Interface Pointer:=Interface Pointer Service Name:=Disconnect List of Unified Service In-Parameter:=(Count, List of pProviderID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
330	ACTIVE	**Disconnect.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=Disconnect hresult:=hresult List of Unified Service Out-Parameter:=(List of ppError) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
331	ACTIVE	**Disconnect.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Disconnect hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
332	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Disconnect => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pProviderID:=List of Unified Service In-Parameter(List of pProviderID) Disonnect.ind(Interface Pointer, Count, List of pProviderID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
333	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Disconnect && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter(List of ppError) Disconnect.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
334	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Disconnect && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Disconnect.cnf(-)(Interface Pointer, hresult)	ACTIVE
335	ACTIVE	**DisconnectMe.req(Interface Pointer, Consumer)** => Interface Pointer:=Interface Pointer Service Name:=DisconnectMe List of Unified Service In-Parameter:=Consumer ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
336	ACTIVE	**DisconnectMe.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer Service Name:=DisconnectMe hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
337	ACTIVE	**DisconnectMe.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=DisconnectMe hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
338	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=DisconnectMe => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter DisonnectMe.ind(Interface Pointer, Consumer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
339	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=DisconnectMe && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult DisconnectMe.cnf(+)(Interface Pointer, hresult)	ACTIVE
340	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=DisconnectMe && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult DisconnectMe.cnf(-)(Interface Pointer, hresult)	ACTIVE
341	ACTIVE	**SetActivation.req(Interface Pointer, State, Count, List of pProviderID)** => Interface Pointer:= Interface Pointer Service Name:=SetActivation List of Unified Service In-Parameter:=(State, Count, List of pProviderID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
342	ACTIVE	**SetActivation.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=SetActivation hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
343	ACTIVE	**SetActivation.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=SetActivation hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
344	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=SetActivation => Interface Pointer:=Interface Pointer State:=List of Unified Service In-Parameter(State) Count:=List of Unified Service In-Parameter(Count) List of pProviderID:=List of Unified Service In-Parameter(List of pProviderID) SetActivation.ind(Interface Pointer, State, Count, List of pProviderID)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
345	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=SetActivation && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter SetActivation.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
346	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=SetActivation && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult SetActivation.cnf(-)(Interface Pointer, hresult)	ACTIVE
347	ACTIVE	**Ping.req(Interface Pointer, Consumer)** => Interface Pointer:=Interface Pointer Service Name:=Ping List of Unified Service In-Parameter:=Consumer ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
348	ACTIVE	**Ping.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Ping hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
349	ACTIVE	**Ping.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Ping hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
350	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Ping => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter Ping.ind(Interface Pointer, Consumer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
351	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Ping && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Ping.cnf(+)(Interface Pointer, hresult)	ACTIVE
352	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Ping && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Ping.cnf(-)(Interface Pointer, hresult)	ACTIVE
353	ACTIVE	**Connect2.req(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn)** => Interface Pointer:=Interface Pointer Service Name:=Connect2 List of Unified Service In-Parameter:=(Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
354	ACTIVE	**Connect2.rsp(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut)** => Interface Pointer:=Interface Pointer Service Name:=Connect2 hresult:=hresult List of Unified Service Out-Parameter:=(pFirstConnect, List of ppConnectOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
355	ACTIVE	**Connect2.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Connect2 hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
356	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Connect2 => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter(Consumer) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) State:=List of Unified Service In-Parameter(State) pICBACallback:=List of Unified Service In-Parameter(pICBACallback) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) Connect2.ind(Interface Pointer, Consumer, QoSType, QoSValue, State, pICBAAccoCallback, Count, List of pConnectIn)	ACTIVE
357	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Connect2 && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pFirstConnect:=List of Unified Service Out-Parameter(pFirstConnect) List of ppConnectOut:=List of Unified Service Out-Parameter(List of ppConnectOut) Connect2.cnf(+)(Interface Pointer, hresult, pFirstConnect, List of ppConnectOut)	ACTIVE
358	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Connect2 && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult Connect2.cnf(-)(Interface Pointer, hresult)	ACTIVE
359	ACTIVE	**GetConnectionData.req(Interface Pointer, Consumer)** => Interface Pointer:=Interface Pointer Service Name:=GetConnectionData List of Unified Service In-Parameter:=Consumer ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
360	ACTIVE	**GetConnectionData.rsp(+)(Interface Pointer, hresult, pLength, pBuffer)** => Interface Pointer:=Interface Pointer Service Name:=GetConnectionData hresult:=hresult List of Unified Service Out-Parameter:=(pLength, pBuffer) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
361	ACTIVE	**GetConnectionData.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=GetConnectionData hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
362	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=GetConnectionData => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter GetConnectionData.ind(Interface Pointer, Consumer)	ACTIVE
363	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConnectionData && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pLength:=List of Unified Service In-Parameter(pLength) pBuffer:=List of Unified Service In-Parameter(pBuffer) GetConnectionData.cnf(+)(Interface Pointer, hresult, pLength, pBuffer)	ACTIVE
364	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=GetConnectionData && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult GetConnectionData.cnf(-)(Interface Pointer, hresult)	ACTIVE
365	ACTIVE	**OnDataChanged.req(Interface Pointer, Length, pBuffer)** => Interface Pointer:=Interface Pointer Service Name:=OnDataChanged List of Unified Service In-Parameter:=(Length, pBuffer) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
366	ACTIVE	**OnDataChanged.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=OnDataChanged hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
367	ACTIVE	**OnDataChanged.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=OnDataChanged hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
368	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=OnDataChanged => Interface Pointer:=Interface Pointer Length:=List of Unified Service In-Parameter(Length) pBuffer:=List of Unified Service In-Parameter(pBuffer) OnDataChanged.ind(Interface Pointer, Length, pBuffer)	ACTIVE
369	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=OnDataChanged && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult OnDataChanged.cnf(+)(Interface Pointer, hresult)	ACTIVE
370	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=OnDataChanged && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult OnDataChanged.cnf(-)(Interface Pointer, hresult)	ACTIVE
371	ACTIVE	**Gnip.req(Interface Pointer)** => Interface Pointer:=Interface Pointer Service Name:=Gnip List of Unified Service In-Parameter:="empty" ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
372	ACTIVE	**Gnip.rsp(+)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Gnip hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
373	ACTIVE	**Gnip.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=Gnip hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
374	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=Gnip => Interface Pointer:=Interface Pointer Gnip.ind(Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
375	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Gnip && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult Gnip.cnf(+)(Interface Pointer, hresult)	ACTIVE
376	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=Gnip && hresult.S=Errors => Interface Pointer:=Interface Pointer hresult:=hresult Gnip.cnf(-)(Interface Pointer, hresult)	ACTIVE
377	ACTIVE	**ReadItems.req(Interface Pointer, Count, List of pReadItem)** => Interface Pointer:=Interface Pointer Service Name:=ReadItems List of Unified Service In-Parameter:=(Count, List of pReadItem) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
378	ACTIVE	**ReadItems.rsp(+)(Interface Pointer, hresult, List of ppReadItemOut)** => Interface Pointer:=Interface Pointer Service Name:=ReadItems hresult:=hresult List of Unified Service Out-Parameter:=List of ppReadItemOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
379	ACTIVE	**ReadItems.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=ReadItems hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
380	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=ReadItems => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pReadItem:=List of Unified Service In-Parameter(List of pReadItem) ReadItems.ind(Interface Pointer, Count, List of pReadItem)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
381	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ReadItems && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppReadItemOut:=List of Unified Service Out-Parameter ReadItems.cnf(+)(Interface Pointer, hresult, List of ppReadItemOut)	ACTIVE
382	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ReadItems && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ReadItems.cnf(-)(Interface Pointer, hresult)	ACTIVE
383	ACTIVE	**WriteItems.req(Interface Pointer, Count, List of pWriteItem)** => Interface Pointer:=Interface Pointer Service Name:=WriteItems List of Unified Service In-Parameter:=(Count, List of pWriteItem) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
384	ACTIVE	**WriteItems.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=WriteItems hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
385	ACTIVE	**WriteItems.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=WriteItems hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
386	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=WriteItems => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pWriteItem:=List of Unified Service In-Parameter(List of pWriteItem) WriteItems.ind(Interface Pointer, Count, List of pWriteItem)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
387	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=WriteItems && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter WriteItems.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
388	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=WriteItems && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult WriteItems.cnf(-)(Interface Pointer, hresult)	ACTIVE
389	ACTIVE	**WriteItemsQCD.req(Interface Pointer, Count, List of pWriteItemQCDIn)** => Interface Pointer:=Interface Pointer Service Name:=WriteItemsQCD List of Unified Service In-Parameter:=(Count, List of pWriteItemQCDIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
390	ACTIVE	**WriteItemsQCD.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=WriteItemsQCD hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
391	ACTIVE	**WriteItemsQCD.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=ReadItemsQCD hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
392	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=WriteItemsQCD => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pWriteItemQCDIn:=List of Unified Service In-Parameter(List of pWriteItemQCDIn) WriteItemsQCD.ind(Interface Pointer, Count, List of pWriteItemQCDIn)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
393	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=WriteItemsQCD && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter WriteItemsQCD.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
394	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=WriteItemsQCD && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult WriteItemsQCD.cnf(-)(Interface Pointer, hresult)	ACTIVE
395	ACTIVE	**ConnectCR.req(Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn)** => Interface Pointer:=Interface Pointer Service Name:=ConnectCR List of Unified Service In-Parameter:=(Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
396	ACTIVE	**ConnectCR.rsp(+)(Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut)** => Interface Pointer:=Interface Pointer Service Name:=ConnectCR hresult:=hresult List of Unified Service Out-Parameter:=(pFirstConnect, pProviderMAC, List of ppConnectOut) ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
397	ACTIVE	**ConnectCR.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=ConnectCR hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
398	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=ConnectCR => Interface Pointer:=Interface Pointer Consumer:=List of Unified Service In-Parameter(Consumer) QoSType:=List of Unified Service In-Parameter(QoSType) QoSValue:=List of Unified Service In-Parameter(QoSValue) pICBACallback:=List of Unified Service In-Parameter(pICBACallback) ConsumerMAC:=List of Unified Service In-Parameter(ConsumerMAC) Flags:=List of Unified Service In-Parameter(Flags) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) ConnectCR.ind(Interface Pointer, Consumer, QoSType, QoSValue, pICBAAccoCallback, ConsumerMAC, Flags, Count, List of pConnectIn)	ACTIVE
399	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ConnectCR && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult pFirstConnect:=List of Unified Service Out-Parameter(pFirstConnect) pProviderMAC:=List of Unified Service Out-Parameter(pProviderMAC) List of ppConnectOut:=List of Unified Service Out-Parameter(List of ppConnectOut) ConnectCR.cnf(+)(Interface Pointer, hresult, pFirstConnect, pProviderMAC, List of ppConnectOut)	ACTIVE
400	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ConnectCR && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult ConnectCR.cnf(-)(Interface Pointer, hresult)	ACTIVE
401	ACTIVE	**DisconnectCR.req(Interface Pointer, Count, List of pProviderCRID)** => Interface Pointer:=Interface Pointer Service Name:=DisconnectCR List of Unified Service In-Parameter:=(Count, List of pProviderCRID) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
402	ACTIVE	**DisconnectCR.rsp(+)(Interface Pointer, hresult, List of ppError)** => Interface Pointer:=Interface Pointer Service Name:=DisconnectCR hresult:=hresult List of Unified Service Out-Parameter:=List of ppError ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
403	ACTIVE	**DisconnectCR.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=DisconnectCR hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
404	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=DisconnectCR => Interface Pointer:=Interface Pointer Count:=List of Unified Service In-Parameter(Count) List of pProviderCRID:=List of Unified Service In-Parameter(List of pProviderCRID) DisconnectCR.ind(Interface Pointer, Count, List of pProviderCRID)	ACTIVE
405	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=DisconnectCR && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppError:=List of Unified Service Out-Parameter DisconnectCR.cnf(+)(Interface Pointer, hresult, List of ppError)	ACTIVE
406	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=DisconnectCR && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult DisconnectCR.cnf(-)(Interface Pointer, hresult)	ACTIVE
407	ACTIVE	**SRTConnect.req(Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn)** => Interface Pointer:=Interface Pointer Service Name:=SRTConnect List of Unified Service In-Parameter:=(ProviderCRID, State, LastConnect, Count, List of pConnectIn) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
408	ACTIVE	**SRTConnect.rsp(+)(Interface Pointer, hresult, List of ppConnectOut)** => Interface Pointer:=Interface Pointer Service Name:=SRTConnect hresult:=hresult List of Unified Service Out-Parameter:=List of ppConnectOut ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
409	ACTIVE	**SRTConnect.rsp(-)(Interface Pointer, hresult)** => Interface Pointer:=Interface Pointer Service Name:=SRTConnect hresult:=hresult List of Unified Service Out-Parameter:="empty" ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
410	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=SRTConnect => Interface Pointer:=Interface Pointer ProviderCRID:=List of Unified Service In-Parameter(ProviderCRID) State:=List of Unified Service In-Parameter(State) LastConnect:=List of Unified Service In-Parameter(LastConnect) Count:=List of Unified Service In-Parameter(Count) List of pConnectIn:=List of Unified Service In-Parameter(List of pConnectIn) SRTConnect.ind(Interface Pointer, ProviderCRID, State, LastConnect, Count, List of pConnectIn)	ACTIVE
411	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=SRTConnect && hresult.S=Success => Interface Pointer:=Interface Pointer hresult:=hresult List of ppConnectOut:=List of Unified Service Out-Parameter SRTConnect.cnf(+)(Interface Pointer, hresult, List of ppConnectOut)	ACTIVE
412	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=SRTConnect && hresult.S=Error => Interface Pointer:=Interface Pointer hresult:=hresult SRTConnect.cnf(-)(Interface Pointer, hresult)	ACTIVE
413	ACTIVE	**CoCreateInstance.req(Host address, Class ID, Interface ID)** => Host address := Host address Class ID := Class ID Interface ID := Interface ID ARPM_CoCreateInstance.req(Host address, Class ID, Interface ID)	ACTIVE
414	ACTIVE	**CoCreateInstance.rsp(+)(hresult, Interface Pointer)** => hresult:=hresult Interface Pointer:= Interface Pointer ARPM_CoCreateInstance.rsp(hresult, Interface Pointer)	ACTIVE
415	ACTIVE	**CoCreateInstance.rsp(-)(hresult)** => hresult:=hresult Interface Pointer:="empty" ARPM_CoCreateInstance.rsp(hresult, Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
416	ACTIVE	**ARPM_CoCreateInstance.ind(Host address, Class ID, Interface ID)** => Host address := Host address Class ID := Class ID Interface ID := Interface ID CoCreateInstance.ind(Host address, Class ID, Interface ID)	ACTIVE
417	ACTIVE	**ARPM_CoCreateInstance.cnf(hresult, Interface Pointer)** /hresult.S=Success => Interface Pointer:= Interface Pointer hresult:=hresult CoCreateInstance.cnf(+)(hresult, Interface Pointer)	ACTIVE
418	ACTIVE	**ARPM_CoCreateInstance.cnf(hresult, Interface Pointer)** /hresult.S=Error => hresult:=hresult CoCreateInstance.cnf(-)(hresult)	ACTIVE
419	ACTIVE	**CoDisconnectObject.req(Interface Pointer)** => Interface Pointer:= Interface Pointer ARPM_CoDisconnectObject.req(Interface Pointer)	ACTIVE
420	ACTIVE	**CoDisconnectObject.rsp(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer hresult:=hresult ARPM_CoDisconnectObject.rsp(Interface Pointer, hresult)	ACTIVE
421	ACTIVE	**ARPM_CoDisconnectObject.ind(Interface Pointer)** => Interface Pointer:= Interface Pointer CoDisconnectObject.ind(Interface Pointer)	ACTIVE
422	ACTIVE	**ARPM_CoDisconnectObject.cnf(Interface Pointer, hresult)** => Interface Pointer:= Interface Pointer hresult:=hresult CoDisconnectObject.cnf(Interface Pointer, hresult)	ACTIVE
423	ACTIVE	**Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)** => Interface Pointer:=Interface Pointer Service Name:=ARPM_Call List of Unified Service In-Parameter:=(Service Name, List of Unified Service In-Parameter) ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
424	ACTIVE	**Call.rsp(+)(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** => Interface Pointer:= Interface Pointer Service Name:=ARPM_Call hresult:=hresult List of Unified Service Out-Parameter:=Service Name, List of Unified Service Out-Parameter ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
425	ACTIVE	**Call.rsp(-)(Interface Pointer, Service Name, hresult)** => Interface Pointer:= Interface Pointer Service Name:=ARPM_Call hresult:=hresult List of Unified Service Out-Parameter:=Service Name ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
426	ACTIVE	**ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)** /Service Name=ARPM_Call => Interface Pointer:=Interface Pointer Service Name=List of Unified Service In-Parameter(Service Name) List of Unified Service In-Parameter:=List of Unified Service In-Parameter(List of Unified Service In-Parameter) Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
427	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ARPM_Call && hresult.S=Success => Interface Pointer:=Interface Pointer Service Name=List of Unified Service Out-Parameter(Service Name) hresult:=hresult List of Unified Service Out-Parameter:=List of Unified Service In-Parameter(List of Unified Service Out-Parameter) Call.cnf(+)(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
428	ACTIVE	**ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** /Service Name=ARPM_Call && hresult.S=Error => Interface Pointer:=Interface Pointer Service Name=List of Unified Service Out-Parameter hresult:=hresult Call.cnf(-)(Interface Pointer, Service Name, hresult)	ACTIVE

5.6 Application relationship protocol machine (ARPM)

5.6.1 Overview

The type 10 specifies one AR endpoint class referred to as ORPC AR. Therefore, one instance of the state machine is specified and the service attribute AREP to address this state machine is omitted.

NOTE Additional ARPM machines can be added in future editions.

The class formal model is specified in IEC 61158-5-10.

5.6.2 Primitive definitions

5.6.2.1 Primitives Exchanged between FSPM and ARPM

The internal service primitives issued by FSPM to ARPM are specified in Table 215.

Table 215 – Primitives issued by FSPM to ARPM

Primitive Names	Source	Associated Parameters	Functions
ARPM_Call.req	FSPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	This primitive is used to request the ARPM to transfer a FAL service request.
ARPM_Call.rsp	FSPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	This primitive is used to request the ARPM to transfer a FAL service response.
ARPM_CoCreateInstance.req	FSPM	Host address, Class ID, Interface ID	This primitive is used to request the ARPM to transfer the FAL CoCreateInstance service request.
ARPM_CoCreateInstance.rsp	FSPM	hresult, Interface Pointer	This primitive is used to request the ARPM to transfer the FAL CoCreateInstance service response.
ARPM_CoDisconnectObject.req	FSPM	Interface Pointer	This primitive is used to request the ARPM to transfer the FAL CoDisconnectObject service request.
ARPM_CoDisconnectObject.rsp	FSPM	Interface Pointer, hresult	This primitive is used to request the ARPM to transfer the FAL CoDisconnectObject service response.

The internal service primitives issued by ARPM to FSPM are specified in Table 216.

Table 216 – Primitives issued by ARPM to FSPM

Primitive Names	Source	Associated Parameters	Functions
ARPM_Call.ind	ARPM	Interface Pointer, Service Name, List of Unified Service In-Parameter	This primitive is used to indicate the FSPM the reception of an ARPM Call.
ARPM_Call.cnf	ARPM	Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter	This primitive is used to indicate the FSPM the reception of an ARPM Call response.
ARPM_CoCreateInstance.ind	ARPM	Host address, Class ID, Interface ID	This primitive is used to indicate the FSPM the reception of an ARPM CoCreateInstance.
ARPM_CoCreateInstance.cnf	ARPM	hresult, Interface Pointer	This primitive is used to indicate the FSPM the reception of an ARPM CoCreateInstance response.
ARPM_CoDisconnectObject.ind	ARPM	Interface Pointer	This primitive is used to indicate the FSPM the reception of an ARPM CoDisconnectObject.
ARPM_CoDisconnectObject.cnf	ARPM	Interface Pointer, hresult	This primitive is used to indicate the FSPM the reception of an ARPM CoDisconnectObject response.

5.6.2.2 Parameters of FSPM/ARPM primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 217.

Table 217 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter Name	Description
Interface Pointer	This parameter specifies the interface address that uniquely represents the associated interface instance.
Service Name	This parameter specifies the service name of the FAL service.
List of Unified Service In-Parameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [in] attribute in one unified container.
List of Unified Service Out-Parameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [out] attribute in one unified container.
Host address	This parameter contains the IP address or DNS name of the node hosting the object to create.
Class ID	This parameter contains the Class ID of the object to create according to its FAL class specification and IDL specification.
Interface ID	This parameter contains the Interface ID of the interface of the created object. It is set according to its FAL class specification and IDL specification.
hresult	This parameter contains the result code.

5.6.2.3 ARPM states

The defined state of the ARPM together with the description is listed in Table 218.

Table 218 – ARPM state descriptions

State Name	Description
ACTIVE	The ARPM in the ACTIVE state is ready to transmit and receive service primitives to and from the DMPM and the FSPM.

The state transition diagram of the ARPM is shown in Figure 46.

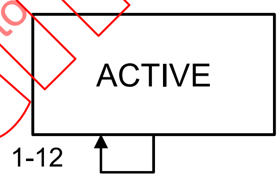

Figure 46 – State transition diagram of ARPM

5.6.2.4 ARPM state table

The ARPM state transitions are specified in Table 219.

Table 219 – ARPM state table

#	Current State	Event /Condition =>Action	Next State
1	ACTIVE	**ARPM_Call.req(Interface Pointer, Service Name, List of Unified Service In-Parameter)** => pInterface:= Interface Pointer serviceName:=Service Name unifiedInParameter:=List of Unified Service In-Parameter referenceToIdl:=IDL Marshaling Reference DMPM_Call.req(pInterface, serviceName, unifiedInParameter, referenceToIdl)	ACTIVE
2	ACTIVE	**ARPM_Call.rsp(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)** => pInterface:= Interface Pointer serviceName:=Service Name hresult:=hresult unifiedOutParameter:=List of Unified Service Out-Parameter referenceToIdl:=IDL Marshaling Reference DMPM_Call.rsp(pInterface, serviceName, hresult, unifiedOutParameter, referenceToIdl)	ACTIVE
3	ACTIVE	**ARPM_CoCreateInstance.req(Host address, Class ID, Interface ID)** => hostAddress:=Host address classID:=Class ID interfaceID:=Interface ID DMPM_Create.req(hostAddress, classID, interfaceID)	ACTIVE
4	ACTIVE	**ARPM_CoCreateInstance.rsp(, hresult, Interface Pointer)** => hresult:=hresult pInterface:= Interface Pointer DMPM_Create.rsp(hresult, pInterface)	ACTIVE
5	ACTIVE	**ARPM_CoDisconnectObject.req(Interface Pointer)** => pInterface:= Interface Pointer DMPM_Disconnect.req(pInterface)	ACTIVE
6	ACTIVE	**ARPM_CoDisconnectObject.rsp(Interface Pointer, hresult)** => pInterface:= Interface Pointer hresult:=hresult DMPM_Disconnect.rsp(pInterface, hresult)	ACTIVE
7	ACTIVE	**DMPM_Call.ind(pInterface, serviceName, unifiedInParameter)** => Interface Pointer:=pInterface Service Name:=serviceName List of Unified Service In-Parameter:=unifiedInParameter ARPM_Call.ind(Interface Pointer, Service Name, List of Unified Service In-Parameter)	ACTIVE
8	ACTIVE	**DMPM_Call.cnf(pInterface, serviceName, hresult, unifiedOutParameter)** => Interface Pointer:=pInterface Service Name:=serviceName hresult:=hresult List of Unified Service Out-Parameter:=unifiedOutParameter ARPM_Call.cnf(Interface Pointer, Service Name, hresult, List of Unified Service Out-Parameter)	ACTIVE
9	ACTIVE	**DMPM_Create.ind(hostAddress, classID, interfaceID)** => Host address:=hostAddress Class ID:=classID Interface ID:=interfaceID ARPM_CoCreateInstance.ind(Host address, Class ID, Interface ID)	ACTIVE
10	ACTIVE	**DMPM_Create.cnf(hresult, pInterface)** => hresult:=hresult Interface Pointer:=pInterface ARPM_CoCreateInstance.cnf(hresult, Interface Pointer)	ACTIVE

#	Current State	Event /Condition =>Action	Next State
11	ACTIVE	**DMPM_Disconnect.ind(pInterface)** => Interface Pointer:=pInterface ARPM_CoDisconnectObject.ind(Interface Pointer)	ACTIVE
12	ACTIVE	**DMPM_Disconnect.cnf(pInterface, hresult)** => Interface Pointer:=pInterface hresult:=hresult ARPM_CoDisconnectObject.cnf(Interface Pointer, hresult)	ACTIVE

5.7 DLL mapping protocol machines (DMPMs)

5.7.1 Overview

The DLL mapping is represented by the ORPC (object oriented remote procedure call) model.

The ORPC model defines an abstract service interface that permits the underlying layer to exist at the data link layer, network layer, or transport layer. It permits, for example, using existing implementations of an object oriented remote procedure.

Additionally, the DLL mapping according 4.16.2 shall be applied for real-time transmission.

5.7.2 Primitive Definitions

5.7.2.1 Primitives Exchanged between DMPM and ARPM

The internal service primitives issued by ARPM to DMPM are specified in Table 220.

Table 220 – Primitives issued by ARPM to DMPM

Primitive Names	Source	Associated Parameters	Functions
DMPM_Call.req	ARPM	pInterface, serviceName, unifiedInParameter, refernceToIdl	This primitive is used to request the DMPM to transfer an ORPC call.
DMPM_Call.rsp	ARPM	pInterface, serviceName, hresult, unifiedOutParameter, refernceToIdl	This primitive is used to request the DMPM to transfer an ORPC response.
DMPM_Create.req	ARPM	hostAddress, classID, interfaceID	This primitive is used to request the DMPM to transfer an ORPC create.
DMPM_Create.rsp	ARPM	hresult, pInterface	This primitive is used to request the DMPM to transfer an ORPC create response.
DMPM_Disconnect.req	ARPM	pInterface	This primitive is used to request the DMPM to transfer an ORPC delete.
DMPM_Disconnect.rsp	ARPM	pInterface, hresult	This primitive is used to request the DMPM to transfer an ORPC delete response.

The internal service primitives issued by DMPM to ARPM are specified in Table 221.

Table 221 – Primitives issued by DMPM to ARPM

Primitive Names	Source	Associated Parameters	Functions
DMPM_Call.ind	DMPM	pInterface, serviceName, unifiedInParameter	This primitive is used to indicate the ARPM the transfer of an ORPC call.
DMPM_Call.cnf	DMPM	pInterface, serviceName, hresult, unifiedOutParameter	This primitive is used to indicate the ARPM the transfer of an ORPC response.
DMPM_Create.ind	DMPM	hostAddress, classID, interfaceID	This primitive is used to indicate the ARPM the transfer of an ORPC create.
DMPM_Create.cnf	DMPM	hresult, pInterface	This primitive is used to indicate the ARPM the transfer of an ORPC create response.
DMPM_Disconnect.ind	DMPM	pInterface	This primitive is used to indicate the ARPM the transfer of an ORPC delete.
DMPM_Disconnect.cnf	DMPM	pInterface, hresult	This primitive is used to indicate the ARPM the transfer of an ORPC delete response.

5.7.2.2 Parameters of ARPM/DMPM Primitives

The parameters used with the primitives exchanged between the ARPM and the DMPM are described in Table 222.

Table 222 – Parameters used with primitives exchanged between ARPM and DMPM

Parameter Name	Description
pInterface	This parameter specifies the interface address that uniquely represents the associated interface instance.
serviceName	This parameter specifies the service name of the FAL service.
unifiedInParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [in] attribute in one unified container.
unifiedOutParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [out] attribute in one unified container.
referenceToIdl	This parameter contains a reference to the IDL specification that shall be used by the abstract ORPC for data marshaling of the unified in and out parameter.
hostAddress	This parameter contains the IP address or DNS name of the node hosting the object to create.
classID	This parameter contains the Class ID of the object to create according to its FAL class specification and IDL specification.
interfaceID	This parameter contains the Interface ID of the object to create according to its FAL class specification and IDL specification.
hresult	This parameter contains the result code.

5.7.2.3 Primitives Exchanged between the ORPC model and DMPM

The internal service primitives issued by DMPM to ORPC model are specified in Table 223.

Table 223 – Primitives issued by DMPM to ORPC model

Primitive Names	Source	Associated Parameters	Functions
ORPC_Send.req	DMPM	pInterface, serviceName, unifiedInParameter, referenceToIdl	This primitive is used to request the ORPC model to transfer an ORPC call.
ORPC_Send.rsp	DMPM	pInterface, serviceName, hresult, unifiedOutParameter, referncеToIdl	This primitive is used to request the ORPC model to transfer an ORPC response.
ORPC_Create.req	DMPM	hostAddress, classID, interfaceID	This primitive is used to request the ORPC model to transfer an ORPC create.
ORPC_Create.rsp	DMPM	hresult, pInterface	This primitive is used to request the ORPC model to transfer an ORPC create response.
ORPC_Delete.req	DMPM	pInterface	This primitive is used to request the ORPC model to transfer an ORPC delete.
ORPC_Delete.rsp	DMPM	pInterface, hresult	This primitive is used to request the ORPC model to transfer an ORPC delete response.

The internal service primitives issued by the ORPC model to DMPM are specified in Table 224.

Table 224 – Primitives issued by ORPC model to DMPM

Primitive Names	Source	Associated Parameters	Functions
ORPC_Send.ind	ORPC model	pInterface, serviceName, unifiedInParameter	This primitive is used to indicate the DMPM the transfer of an ORPC call.
ORPC_Send.cnf	ORPC model	pInterface, serviceName, hresult, unifiedOutParameter	This primitive is used to indicate the DMPM the transfer of an ORPC response.
ORPC_Create.ind	ORPC model	hostAddress, classID, interfaceID	This primitive is used to indicate the DMPM the transfer of an ORPC create.
ORPC_Create.cnf	ORPC model	hresult, pInterface	This primitive is used to indicate the DMPM the transfer of an ORPC create response.
ORPC_Delete.ind	ORPC model	pInterface	This primitive is used to indicate the DMPM the transfer of an ORPC delete.
ORPC_delete.cnf	ORPC model	pInterface, hresult	This primitive is used to indicate the DMPM the transfer of an ORPC delete response.

5.7.2.4 Parameters of DMPM/ORPC model primitives

The parameters used with the primitives exchanged between the DMPM and the ORPC model are specified in Table 225.

Table 225 – Parameters used with primitives exchanged between DMPM and ORPC model

Parameter Name	Description
pInterface	This parameter specifies the interface address that uniquely represents the associated interface instance.
serviceName	This parameter specifies the service name of the FAL service.
unifiedInParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [in] attribute in one unified container.
unifiedOutParameter	This parameter summarizes all FAL service parameters specified in the related IDL with the [out] attribute in one unified container.
referenceToIdl	This parameter contains a reference to the IDL specification that shall be used by the abstract ORPC for data marshaling of the unified in and out parameter.
hostAddress	This parameter contains the IP address or DNS name of the node hosting the object to create.
classID	This parameter contains the Class ID of the object to create according to its FAL class specification and IDL specification.
interfaceID	This parameter contains the Interface ID of the object to create according to its FAL class specification and IDL specification.
hresult	This parameter contains the result code.

5.7.2.5 DMPM states

The defined state of the DMPM together with the description is listed in Table 226.

Table 226 – DMPM state descriptions

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

The state transition diagram of the DMPM is shown in Figure 47.

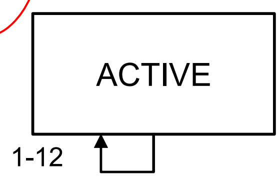

Figure 47 – State transition diagram of DMPM

5.7.2.6 DMPM state table

The DMPM state transitions are specified in Table 227.

Table 227 – DMPM state table

#	Current State	Event /Condition =>Action	Next State
1	ACTIVE	**DMPM_Call.req(pInterface, serviceName, unifiedInParameter, referenceToIdl)** => ORPC_Send.req(pInterface, serviceName, unifiedInParameter, referenceToIdl)	ACTIVE
2	ACTIVE	**DMPM_Call.rsp(pInterface, serviceName, hresult, unifiedOutParameter, referenceToIdl)** => ORPC_Send.rsp(pInterface, serviceName, hresult, unifiedOutParameter, referenceToIdl)	ACTIVE
3	ACTIVE	**DMPM_Create.req(hostAddress, classID, interfaceID)** => ORPC_Create.req(hostAddress, classID, interfaceID)	ACTIVE
4	ACTIVE	**DMPM_Create.rsp(hresult, pInterface)** => ORPC_Create.rsp(hresult, pInterface)	ACTIVE
5	ACTIVE	**DMPM_Disconnect.req(pInterface)** => ORPC_Disconnect.req(pInterface)	ACTIVE
6	ACTIVE	**DMPM_Disconnect.rsp(pInterface, hresult)** => ORPC_Disconnect.rsp(pInterface, hresult)	ACTIVE
7	ACTIVE	**ORPC_Send.ind(pInterface, serviceName, unifiedInParameter)** => DMPM_Call.ind(pInterface, serviceName, unifiedInParameter)	ACTIVE
8	ACTIVE	**ORPC_Send.cnf(pInterface, serviceName, hresult, unifiedOutParameter)** => DMPM_Call.cnf(pInterface, serviceName, hresult, unifiedOutParameter)	ACTIVE
9	ACTIVE	**ORPC_Create.ind(hostAddress, classID, interfaceID)** => DMPM_Create.ind(hostAddress, classID, interfaceID)	ACTIVE
10	ACTIVE	**ORPC_Create.cnf(hresult, pInterface)** => DMPM_Create.cnf(hresult, pInterface)	ACTIVE
11	ACTIVE	**ORPC_Disconnect.ind(pInterface)** => DMPM_Disconnect.ind(pInterface)	ACTIVE
12	ACTIVE	**ORPC_Disconnect.cnf(pInterface, hresult)** => DMPM_Disconnect.cnf(pInterface, hresult)	ACTIVE

5.8 Protocol options

Type 10 has no protocol options.

6 Application layer protocol specification for decentralized periphery

6.1 FAL syntax description

6.1.1 DLPDU abstract syntax reference

The DLPDU abstract syntax from 4.1.1 shall be applied.

6.1.2 APDU abstract syntax

Table 228 defines the abstract syntax of the Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs. The APDUs of Table 228 shall represent the content of the DLSDU in Table 4.

Table 228 – IO APDU substitutions

Substitution name	Structure
RTA-SDU	AlarmNotification-PDU ^ AlarmAck-PDU
BlockHeader	BlockType, BlockLength, BlockVersionHigh, BlockVersionLow
AlarmNotification-PDU	BlockHeader, AlarmType, API, SlotNumber, SubslotNumber, ModuleIdentNumber, SubmoduleIdentNumber, AlarmSpecifier, [MaintenanceItem], [AlarmItem] ^ [Upload&Retrieval-Item] ^ [iParameterItem] The BlockType field within the BlockHeader shall be set to AlarmNotification high or low according Table 229.
AlarmAck-PDU	BlockHeader, AlarmType, API, SlotNumber, SubslotNumber, AlarmSpecifier, PNIOStatus The BlockType field within the BlockHeader shall be set to AlarmAck high or low according Table 229. Special case “No Error”: PNIOStatus(=0x00, 0x00, 0x00, 0x00) Special case “Alarm Type Not Supported” or if a reserved Alarm Type is used: PNIOStatus(=0xDA, 0x81, 0x3C, 0x00) Special case “Wrong Submodule State”: PNIOStatus(=0xDA, 0x81, 0x3C, 0x01)
MaintenanceItem	UserStructureIdentifier, BlockHeader, Padding, Padding, MaintenanceStatus
Upload&Retrieval-Item	UserStructureIdentifier, BlockHeader, Padding, Padding, (URRecordIndex, URRecordLength)* Special case “Retrieval all stored records”: UserStructureIdentifier, BlockHeader(=0x0F04), Padding, Padding, URRecordIndex(=0), URRecordLength(=0)
iParameterItem	UserStructureIdentifier, BlockHeader, Padding, Padding, iPar_Req_Header, Max_Segm_Size, Transfer_Index, Total_iPar_Size
AlarmItem	UserStructureIdentifier, Data* ^ ChannelDiagnosisData* ^ DiagnosisData* ExtChannelDiagnosisData* ^ QualifiedChannelDiagnosisData* Special case ChannelDiagnosisData: UserStructureIdentifier(=0x8000), ChannelDiagnosisData* Special case Multiple for AlarmType Diagnosis: UserStructureIdentifier(=0x8001), DiagnosisData* Special case ExtChannelDiagnosisData: UserStructureIdentifier(=0x8002), ExtChannelDiagnosisData* Special case QualifiedChannelDiagnosisData: UserStructureIdentifier(=0x8003), QualifiedChannelDiagnosisData*
PROFINETIO-ServiceReqPDU	IODConnectReq ^ IODWriteReq ^ IODWriteMultipleReq ^ IODReadReq ^ IODControlReq ^ IODReleaseReq ^ IOXControlReq
PROFINETIO-ServiceResPDU	IODConnectRes ^ IODWriteRes ^ IODWriteMultipleRes ^ IODReadRes ^ IODControlRes ^ IODReleaseRes ^ IOXControlRes

Substitution name	Structure
IODConnectReq	ARBlockReq, {[IOCRBlockReq*], [AlarmCRBlockReq], [ExpectedSubmoduleBlockReq*] [b], [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq] } Special case: “IOCAR” or “IOSAR”: ARBlockReq, {IOCRBlockReq*, AlarmCRBlockReq, ExpectedSubmoduleBlockReq*, [PrmServerBlock], [MCRBlockReq*] [a], [ARRPCBlockReq] } Special case: “IOSAR with ARProperties.DeviceAccess:=1”: ARBlockReq [c] [a] The field MCRBlockReq shall only be present if at least one IOCRBlockReq contains the value MULTICAST_CONSUMER_CR as an IO CR Type. [b] The field ExpectedSubmoduleBlockReq shall only contain submodules which are referred in at least one IOCRBlockReq. [c] The field CMInitiatorActivityTimeoutFactor specifies the watchdog time.
IODConnectRes	ARBlockRes, {[IOCRBlockRes*], [AlarmCRBlockRes], [ModuleDiffBlock], [ARRPCBlockRes] [a]} Special case: “IOCAR” or “IOSAR”: ARBlockRes, {IOCRBlockRes*, AlarmCRBlockRes, [ModuleDiffBlock], [ARRPCBlockRes] [a]} Special case: “IOSAR with ARProperties.DeviceAccess=1”: ARBlockRes [a] The field ARRPCBlockRes shall only be present if IODConnectReq contains an ARRPCBlockReq.
IODWriteMultiple-Req	IODWriteReqHeader [a], (IODWriteReqHeader, RecordDataWrite, [Padding*] [b])* [a] The value of the parameter index shall be 0xE040. [b] The number of padding octets (value=0) shall be 0,1,2,3 to have 32 bit alignment to the next IODWriteReqHeader.
IODWriteReq	IODWriteReqHeader, RecordDataWrite
IODWriteReqHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, RWPadding* [b] [a] The number of padding octets (value=0) in the write request is 2. [b] The number of padding octets (value=0) in the write request is 24.
IODWriteRes	IODWriteResHeader
IODWriteMultipleRes	IODWriteResHeader [a], (IODWriteResHeader)* [a] The value of the parameter index shall be 0xE040.
IODWriteResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, AdditionalValue1, AdditionalValue2, PNIOStatus [b], RWPadding* [c] [a] The number of padding octets (value=0) in the write response is 2. [b] In case of IODWriteRes PNIOStatus (=0,0,0,0). [c] The number of padding octets (value=0) in the write response is 16.
IODReadReq	IODReadReqHeader, [RecordDataReadQuery]

Substitution name	Structure
IODReadReqHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, [TargetARUUID] [b], RWPadding* [c] [a] The number of padding octets (value=0) in the read request is 2. [b] The optional field TargetARUUID shall only be used in conjunction with the value 0 of ARUUID (Implicit AR). [c] The number of padding octets (value=0) in the read request is 24 or 8.
RecordDataRead-Query	BlockHeader, Data* [a] [a] The usage, structure and values shall be defined for a) Manufacturer specific index range by the manufacturer b) Profile specific index ranges by the profiles c) All other index ranges by this standard
IODReadRes	IODReadResHeader, RecordDataRead
IODReadResHeader	BlockHeader, SeqNumber, ARUUID, API, SlotNumber, SubslotNumber, Padding* [a], Index, RecordDataLength, AdditionalValue1, AdditionalValue2, RWPadding* [b] [a] The number of padding octets (value=0) in the read res is 2. [b] The number of padding octets (value=0) in the read res is 20.
IODControlReq	ControlBlockConnect ^ ControlBlockPlug
IODControlRes	ControlBlockConnect ^ ControlBlockPlug ^ NULL [a] [a] In case of a negative response NULL shall be transmitted.
IODReleaseReq	ReleaseBlock
IODReleaseRes	ReleaseBlock ^ NULL [a] [a] In case of a negative response NULL shall be transmitted.
IOXControlReq	(ControlBlockConnect, [ModuleDiffBlock]) ^ (ControlBlockPlug, [ModuleDiffBlock]) ^ ControlBlockRFC The field ModuleDiffBlock shall only be present if an Application Ready has to signal an error.
IOXControlRes	ControlBlockConnect ^ ControlBlockPlug ^ NULL [a] [a] In case of a negative response NULL shall be transmitted.
ARBlockReq	BlockHeader, ARType, ARUUID, SessionKey, CMInitiatorMacAdd, CMInitiatorObjectUUID, ARProperties, CMInitiatorActivityTimeoutFactor, InitiatorUDPRTPort, StationNameLength, CMInitiatorStationName The BlockType field within the BlockHeader shall be set to ARBlockReq according Table 229.
ARBlockRes	BlockHeader, ARType, ARUUID, SessionKey, CMResponderMacAdd, ResponderUDPRTPort The BlockType field within the BlockHeader shall be set to ARBlockRes according Table 229.

Substitution name	Structure
IOCRBlockReq	BlockHeader, IOCRType, IOCRReference, LT, IOCRProperties, DataLength, FrameID [a], SendClockFactor, ReductionRatio, Phase, Sequence, FrameSendOffset, WatchdogFactor, DataHoldFactor, IOCRTagHeader, IOCRMulticastMACAdd, NumberOfAPIs, (API, NumberOfIODataObjects, (SlotNumber, SubslotNumber, IODataObjectFrameOffset)*, NumberOfIOCS, (SlotNumber, SubslotNumber, IOCSFrameOffset)*)* The BlockType field within the BlockHeader shall be set to IOCRBlockReq according Table 229. [a] The field FrameID shall be don't care if the field IOCRType contains the value Output CR. Special Case IOCRType = MULTICAST_CONSUMER_CR The field FrameID shall contain a FrameID from the defined multicast range. Theconfiguration tool shall guarantee that this FrameID is unique within the project context. Special Case IOCRType = OUTPUT_CR and IOCRProperties = RT_CLASS_3 The field FrameID shall contain the FrameID used in the correspondig PDIRData.
MCRBlockReq	BlockHeader, IOCRReference, AddressResolutionProperties, MCITimeoutFactor, StationNameLength, ProviderStationName, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
IOCRBlockRes	BlockHeader, IOCRType, IOCRReference, FrameID The BlockType field within the BlockHeader shall be set to IOCRBlockRes according Table 229.
ExpectedSubmodule BlockReq	BlockHeader, NumberOfAPIs, (API, SlotNumber [a], ModuleIdentNumber, ModuleProperties, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber, SubmoduleProperties [b], (DataDescription, SubmoduleDataLength, LengthIOPS, LengthIOCS)*)*)* The BlockType field within the BlockHeader shall be set to ExpectedSubmoduleBlockReq according Table 229. [a] The value of this parameter shall be identical for all APIs for one ExpectedSubmoduleBlockReq. [b] The field SubmoduleProperties.Type determines the number of subsequent data description blocks.
ModuleDiffBlock	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, ModuleState [a], NumberOfSubmodules, [(SubslotNumber, SubmoduleIdentNumber, SubmoduleState)*])*)* [a] If ModuleState=NO_MODUL then NumberOfSubmodules shall be zero. The subsequent part shall be omitted. For all other ModuleState only the SubmoduleState shall be used to decide whether the submodule shall be parametrized. The BlockType field within the BlockHeader shall be set to ModuleDiffBlock according Table 229.
AlarmCRBlockReq	BlockHeader, AlarmCRType, LT, AlarmCRProperties, RTATimeoutFactor, RTARetries, LocalAlarmReference, MaxAlarmDataLength, AlarmCRTagHeaderHigh, AlarmCRTagHeaderLow The BlockType field within the BlockHeader shall be set to AlarmCRBlockReq according Table 229.
AlarmCRBlockRes	BlockHeader, AlarmCRType, LocalAlarmReference, MaxAlarmDataLength The BlockType field within the BlockHeader shall be set to AlarmCRBlockRes according Table 229.

Substitution name	Structure
PrmServerBlock	BlockHeader, ParameterServerObjectUUID, ParameterServerProperties, CMInitiatorActivityTimeoutFactor, StationNameLength, ParameterServerStationName The BlockType field within the BlockHeader shall be set to PrmServerBlock according Table 229.
ARRPCBlockReq	BlockHeader, InitiatorRPCServerPort
ARRPCBlockRes	BlockHeader, ResponderRPCServerPort
ControlBlock-Connect	BlockHeader, Padding* [a], ARUUID, SessionKey, Padding* [a], ControlCommand, ControlBlockProperties [a] The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to IODBlockReq, IODBlockRes, IOXBlockReq and, IOXBlockRes according Table 229.
ControlBlockPlug	BlockHeader, Padding* [a], ARUUID, SessionKey, AlarmSequenceNumber, ControlCommand, ControlBlockProperties [a] The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to IOXBlockReq, IOXBlockRes according Table 229. The field AlarmSequenceNumber shall contain the value of the sub-field AlarmSpecifier.SequenceNumber of the corresponding AlarmNotification-PDU.
ControlBlockRFC	BlockHeader, Padding* [a], ARUUID, SessionKey, Padding* [a], ControlCommand, ControlBlockProperties [a] The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to ReadyForCompanionBlock according Table 229.
ReleaseBlock	BlockHeader, Padding* [a], ARUUID, SessionKey, Padding* [a], ControlCommand, ControlBlockProperties [a] The number of Padding octets shall be 2. The BlockType field within the BlockHeader shall be set to ReleaseBlock according Table 229.
RecordDataRead	DiagnosisData* ^ ExpectedIdentificationData* ^ RealIdentificationData* ^ SubstituteValue ^ RecordInputDataObjectElement ^ RecordOutputDataObjectElement ^ Data* ^ ARData ^ IMData ^ LogData ^ ModuleDiffBlock ^ APIData ^ PDPortDataAdjust* ^ PDPortDataCheck* ^ PDPortDataReal* ^ PDIRData ^ PDSyncData* ^ PDevData ^ IsochronousModeData* ^ PDInterfaceMrpDataAdjust* ^ PDInterfaceMrpDataCheck* ^ PDInterfaceMrpDataReal* ^ PDPortMrpDataAdjust ^ PDPortMrpDataReal ^ PDPortFODataReal ^ PDPortFODataAdjust ^ PDPortFODataCheck ^ PDRealData* ^ PDExpectedData* ^ PDNCDataCheck ^ I&M0FilterData ^ FastStartUp ^ PDInterfaceDataReal ^ NULL [a] [a] NULL shall be used if a requested well-known data record is empty (e.g. Diagnosis, ModuleDiffBlock, ARData, …).
RecordDataWrite	SubstituteValue ^ Data* ^ IMDataWrite ^ PDPortDataAdjust ^ PDPortDataCheck ^ PDIRData ^ PDSyncData ^ IsochronousModeData ^ PDInterfaceMrpDataAdjust ^ PDInterfaceMrpDataCheck ^ PDPortMrpDataAdjust ^ PDPortFODataAdjust ^ PDPortFODataCheck ^ PDNCDataCheck ^ FastStartUp
DiagnosisData with BlockVersionLow=0	BlockHeader, ChannelDiagnosis ^ ManufacturerSpecificDiagnosis ^ ExtChannelDiagnosis BlockVersionLow=0 shall be supported by IO controller and IO supervisor. It shall not be generated by IO device.

Substitution name	Structure
DiagnosisData with BlockVersionLow=1	BlockHeader, API, ChannelDiagnosis ^ ManufacturerSpecificDiagnosis ^ ExtChannelDiagnosis ^ QualifiedChannelDiagnosis BlockVersionLow=1 shall be generated by IO device and support by IO controller and IO supervisor.
ChannelDiagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a], UserStructureIdentifier(0x8000), ChannelDiagnosisData* [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Maintenance shall be set to zero. The field ChannelProperties.Specifier shall be set to appear if at least one ChannelProperties.Specifier in the ChannelDiagnosisData is set to appear in conjunction with ChannelProperties.Maintenance(=diagnosis). Else, the field ChannelProperties.Specifier shall be set to disappear.
ChannelDiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType
Manufacturer-SpecificDiagnosis	SlotNumber, SubslotNumber, ChannelNumber, ChannelProperties, UserStructureIdentifier, Data* The BlockType field within the BlockHeader shall be set to ManufacturerSpecificDiagnosis according Table 229.
ExtChannel-Diagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a], UserStructureIdentifier(0x8002), ExtChannelDiagnosisData * [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Maintenance shall be set to zero. The field ChannelProperties.Specifier shall be set to appear if at least one ChannelProperties.Specifier in the ExtChannelDiagnosisData is set to appear in conjunction with ChannelProperties.Maintenance(=diagnosis). Else, the field ChannelProperties.Specifier shall be set to disappear.
ExtChannel-DiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType, ExtChannelErrorType, ExtChannelAddValue
QualifiedChannel-Diagnosis	SlotNumber, SubslotNumber, ChannelNumber(0x8000), ChannelProperties [a], UserStructureIdentifier(0x8003), QualifiedChannelDiagnosisData * [a] The field ChannelProperties.Type, the field ChannelProperties.Direction, the field ChannelProperties.Maintenance shall be set to zero. The field ChannelProperties.Specifier shall be set to appear if at least one ChannelProperties.Specifier in the QualifiedChannelDiagnosisData is set to appear in conjunction with ChannelProperties.Maintenance(=diagnosis). Else, the field ChannelProperties.Specifier shall be set to disappear.
QualifiedChannel-DiagnosisData	ChannelNumber, ChannelProperties, ChannelErrorType, ExtChannelErrorType, ExtChannelAddValue, QualifiedChannelQualifier
ExpectedIdentificationData with BlockVersionLow = 0	BlockHeader, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber)*)*
ExpectedIdentificationData with BlockVersionLow =1	BlockHeader, NumberOfAPIs, (API, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber)*)*)*
RealIdentificationData with BlockVersionLow = 0	BlockHeader, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber)*)*
RealIdentificationData with BlockVersionLow = 1	BlockHeader, NumberOfAPIs, (API, NumberOfSlots, (SlotNumber, ModuleIdentNumber, NumberOfSubslots, (SubslotNumber, SubmoduleIdentNumber)*)*)*
PDIRData	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, PDIRGlobalData, PDIRFrameData

Substitution name	Structure
PDSyncData with Block-VersionLow = 0	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, PTCPSubdomainID, IRDataUUID [a], ReservedIntervalBegin, ReservedIntervalEnd, PLLWindow, SyncSendFactor, SendClockFactor, SyncProperties, SyncFrameAddress, PTCPTimeoutFactor [a] This field shall only be valid if the field SyncFrameAddress.MulticastSelection contains the value 0.
PDSyncData with Block-VersionLow = 1	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, PTCPSubdomainID, IRDataUUID [b], ReservedIntervalBegin, ReservedIntervalEnd, PLLWindow, SyncSendFactor, SendClockFactor, SyncProperties, SyncFrameAddress, PTCPTimeoutFactor, PTCPLengthSubdomainName, PTCPSubdomainName, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] This field shall only be valid if the field SyncFrameAddress.MulticastSelection contains the value 0.
PDevData	BlockHeader, Padding, Padding, [PDIRData], [PDSyncData*]
PDRealData	MultipleBlockHeader, { [PDPortDataReal] [b], [PDInterfaceMrpDataReal], [PDPortMrpDataReal], [PDPortFODataReal] [a], [PDInterfaceDataReal] } [a] There shall be no FiberOpticManufacturerSpecific information [b] The fields Slotnumber and Subslotnumber shall be ignored
PDExpectedData	MultipleBlockHeader, { [PDPortDataCheck] [a], [PDPortDataAdjust] [a], [PDInterfaceMrpDataAdjust], [PDInterfaceMrpDataCheck], [PDPortMrpDataAdjust], [PDPortFODataAdjust], [PDPortFODataCheck], [PDNCDataCheck] } [a] The fields Slotnumber and Subslotnumber shall be ignored
PDPortDataReal	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, LengthOwnPortID, OwnPortID, NumberOfPeers, [Padding*] [a], [LengthPeerPortID, PeerPortID, LengthPeerChassisID, PeerChassisID, [Padding*] [a], LineDelay, PeerMACAddress [b]]*, [Padding*] [a], MAUType [c], [Padding*] [a], DomainBoundary, MulticastBoundary, PortState, [Padding*] [a], MediaType [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. b This field contains the interface MAC address of the peer [c] See Table 364
PDInterface-DataReal	BlockHeader, LengthOwnChassisID, OwnChassisID, [Padding*] [a], MACAddressValue [b], [Padding*] [a], IPParameterValue, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned. [b] This field contains the interface MAC address
TxPortGroup	NumberOfTxPortGroups, TxPortGroupArray
SubstituteValue	BlockHeader, SubstitutionMode, SubstituteDataItem
RecordInputDataObjectElement	BlockHeader, LengthIOCS, IOCS, LengthIOPS, IOPS, LengthData, Data Special case: Response for an output submodul PNIOStatus(=0xDE, 0x80, 0xB0, 0x00)
RecordOutputDataObjectElement	BlockHeader, SubstituteActiveFlag, LengthIOCS, LengthIOPS, LengthData, DataItem [a], SubstituteValue Special case: Response for an input submodul PNIOStatus(=0xDE, 0x80, 0xB0, 0x00) [a] For this record DataItem shall be coded as IOCS, Data, IOPS

Substitution name	Structure
ARData	BlockHeader, NumberOfARs, (ARUUID, ARType, ARProperties, CMInitiatorObjectUUID, StationNameLength, CMInitiatorStationName, NumberOfIOCRs, (IOCRType, IOCRProperties, FrameID, APDU_Status [a], InitiatorUDPRTPort, ResponderUDPRTPort)*, AlarmCRType, LocalAlarmReference, RemoteAlarmReference, ParameterServerObjectUUID [b], StationNameLength [c], [ParameterServerStationName], NumberOfAPIs, API*)* Special case: "IOSAR with ARProperties.DeviceAccess=1": NumberOfIOCRs := 0 AlarmCRType := 0 NumberOfAPIs := 0 [a] The APDU_Status.CycleCounter and APDU_Status.TransferStatus can be zero [b] The ParameterServerObjectUUID shall be NIL if not used [c] The StationNameLength shall be 0 if not used. In this case the field ParameterServerStationName shall be omitted
APIData	BlockHeader, NumberOfAPIs, API*
LogData	BlockHeader, ActualLocalTimeStamp, NumberOfLogEntries, (LocalTimeStamp, ARUUID, PNIOStatus, EntryDetail)*
CMInitiator-ObjectUUID	PROFINETIOConstantValue, InstanceHigh, InstanceLow, DeviceIdentNumber
ParameterServer-ObjectUUID	PROFINETIOConstantValue, InstanceHigh, InstanceLow, DeviceIdentNumber
DeviceIdentNumber	DeviceIDHigh, DeviceIDLow, VendorIDHigh, VendorIDLow
IMDataWrite	[I&M1] ^ [I&M2] ^ [I&M3] ^ [I&M4]
IMData	I&M0 ^ [I&M1] ^ [I&M2] ^ [I&M3] ^ [I&M4]
I&M0	BlockHeader, VendorIDHigh, VendorIDLow, OrderID, IM_Serial_Number, IM_Hardware_Revision, IM_Software_Revision, IM_Revision_Counter, IM_Profile_ID, IM_Profile_Specific_Type, IM_Version, IM_Supported
I&M1	BlockHeader, IM_Tag_Function, IM_Tag_Location
I&M2	BlockHeader, IM_Date
I&M3	BlockHeader, IM_Descriptor
I&M4	BlockHeader, IM_Signature
I&M0FilterData	{I&M0FilterDataSubmodul [a], [I&M0FilterDataModul] [b], [I&M0FilterDataDevice] [c]} [a] Shall contain all submodules with discrete IMData [b] Shall, if exists, contain only the module reference [c] Shall, if exists, contain only the device reference
I&M0FilterData-Submodul	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*)*)*
I&M0FilterData-Modul	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*)*)*
I&M0FilterData-Device	BlockHeader, NumberOfAPIs, (API, NumberOfModules, (SlotNumber, ModuleIdentNumber, NumberOfSubmodules, (SubslotNumber, SubmoduleIdentNumber)*)*)*
IM_Version	IM_Version_Major, IM_Version_Minor
IM_Software_Revision	SWRevisionPrefix, IM_SWRevision_Functional_Enhancement, IM_SWRevision_Bug_Fix, IM_SWRevision_Internal_Change
MultipleBlockHeader	BlockHeader, Padding, Padding, API, SlotNumber, SubslotNumber
PDIRGlobalData with BlockVersionLow = 0	BlockHeader, Padding, Padding, IRDataUUID

Substitution name	Structure
PDIRGlobalData with BlockVersionLow = 1	BlockHeader, Padding, Padding, IRDataUUID, MaxBridgeDelay, NumberOfPorts, (MaxPortTxDelay, MaxPortRxDelay)*
PDIRFrameData	BlockHeader, Padding, Padding, (FrameSendOffset, DataLength, ReductionRatio, Phase, FrameID, Ethertype, RxPort, FrameDetails, TxPortGroup, [Padding*] [a])* [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
IsochronousMode-Data	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, ControllerApplicationCycleFactor, TimeDataCycle, TimeIOInput, TimeIOOutput, TimeIOInputValid, TimeIOOutputValid
PDPortDataAdjust	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, { [AdjustDomainBoundary], [AdjustMulticastBoundary], [AdjustMAUType ^ AdjustPortState] }
PDPortDataCheck	BlockHeader, Padding, Padding, SlotNumber, SubslotNumber, { [CheckPeers], [CheckLineDelay], [CheckMAUType], [CheckPortState], [CheckSyncDifference], [CheckMAUTypeDifference] }
AdjustDomainBoundary	BlockHeader, Padding, Padding, DomainBoundary, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustMulticastBoundary	BlockHeader, Padding, Padding, MulticastBoundary, AdjustProperties, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
AdjustMAUType	BlockHeader, Padding, Padding, MAUType, AdjustProperties
AdjustPortState	BlockHeader, Padding, Padding, PortState, AdjustProperties
CheckPeers	BlockHeader, NumberOfPeers, (LengthPeerPortID, PeerPortID, LengthPeerChassisID, PeerChassisID)*, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
CheckLineDelay	BlockHeader, Padding, Padding, LineDelay
CheckMAUType	BlockHeader, MAUType
CheckPortState	BlockHeader, PortState
CheckSyncDifference	BlockHeader, CheckSyncMode
CheckMAUType-Difference	BlockHeader, MAUTypeMode
PDInterface-MrpDataAdjust	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, [Padding*] [a], MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], { [(MrpManagerParams, [MrpRTModeManagerData]) ^ (MrpClientParams, [MrpRTModeClientData])] } [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDInterface-MrpDataReal	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Role, MRP_LengthDomainName, MRP_DomainName, [Padding*] [a], MRP_Version, { [(MrpManagerParams, [MrpRTModeManagerData]) ^ (MrpClientParams, [MrpRTModeClientData])], [MrpRingStateData], [MrpRTStateData] }
PDInterface-MrpDataCheck	BlockHeader, Padding, Padding, MRP_DomainUUID, MRP_Check
PDPort-MrpDataAdjust	BlockHeader, Padding, Padding, MRP_DomainUUID
PDPortMrpDataReal	BlockHeader, Padding, Padding, MRP_DomainUUID
MrpManagerParams	BlockHeader, MRP_Prio, MRP_TOPchgT, MRP_TOPNRmax, MRP_TSTshortT, MRP_TSTdefaultT, MRP_TSTNRmax, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MrpClientParams	BlockHeader, MRP_LNKdownT, MRP_LNKupT, MRP_LNKNRmax
MrpRTMode-ManagerData	BlockHeader, MRRT_TSTNRmax, MRRT_TSTdefaultT, [Padding*] [a], MRP_RTMode [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

Substitution name	Structure
MrpRTModeClientData	BlockHeader, [Padding*] [a], MRP_RTMode [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
MrpRingStateData	BlockHeader, MRP_RingState
MrpRTStateData	BlockHeader, MRP_RTState
PDPortFODataReal	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType, [FiberOpticManufacturerSpecific*]
FiberOpticManufacturerSpecific	BlockHeader, VendorIDHigh, VendorIDLow, VendorBlockType, Data*, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
PDPortFODataAdjust	BlockHeader, Padding, Padding, FiberOpticType, FiberOpticCableType
PDPortFODataCheck	BlockHeader, Padding, Padding, MaintenanceRequiredPowerBudget, MaintenanceDemandedPowerBudget, ErrorPowerBudget
Maintenance-Required-PowerBudget	FiberOpticPowerBudgetType
MaintenanceDemandedPowerBudget	FiberOpticPowerBudgetType
ErrorPowerBudget	FiberOpticPowerBudgetType
PDNCDataCheck	BlockHeader, Padding, Padding, MaintenanceRequiredDropBudget, MaintenanceDemandedDropBudget, ErrorDropBudget
MaintenanceRequiredDropBudget	NCDropBudgetType
MaintenanceDemandedDropBudget	NCDropBudgetType
ErrorDropBudget	NCDropBudgetType
FastStartUp	{ [FSHelloBlock], [FSParameterBlock], [FSModeBlock], [FastStartUpBlock] } [a] [a] At least one optional block shall be existing.
FSHelloBlock	BlockHeader, [Padding*] [a], FSHelloMode, FSHelloInterval, FSHelloRetry, FSHelloDelay, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FSParameterBlock	BlockHeader, [Padding*] [a], FSParameterMode, FSParameterUUID, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FSModeBlock	BlockHeader, [Padding*] [a], FSMode, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.
FastStartUpBlock	BlockHeader, [Padding*] [a], Data*, [Padding*] [a] [a] The number of padding octets shall be adapted to make the block Unsigned32 aligned.

6.2 Transfer syntax

6.2.1 Coding section related to BlockHeader specific fields

6.2.1.1 Coding of the field BlockType

This field shall be coded as data type Unsigned16 with the values according to Table 229. This field identifies the PDU or data block type.

Table 229 – BlockType

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	Not used
0x0001	AlarmNotification High	AlarmNotification-PDU
0x0002	AlarmNotification Low	AlarmNotification-PDU
0x0008	IODWriteReqHeader	RecordDataWrite
0x8008	IODWriteResHeader	RecordDataWrite
0x0009	IODReadReqHeader	RecordDataRead
0x8009	IODReadResHeader	RecordDataRead
0x0010	DiagnosisData	RecordDataRead, AlarmNotification-PDU
0x0012	ExpectedIdentificationData	RecordDataRead
0x0013	RealIdentificationData	RecordDataRead
0x0014	SubstituteValue	RecordDataRead, RecordDataWrite
0x0015	RecordInputDataObjectElement	RecordDataRead
0x0016	RecordOutputDataObjectElement	RecordDataRead
0x0017	Reserved	
0x0018	ARData	RecordDataRead
0x0019	LogData	RecordDataRead
0x001A	APIData	RecordDataRead
0x0020	I&M0	RecordDataRead
0x0021	I&M1	RecordDataRead RecordDataWrite
0x0022	I&M2	RecordDataRead RecordDataWrite
0x0023	I&M3	RecordDataRead RecordDataWrite
0x0024	I&M4	RecordDataRead RecordDataWrite
0x0025 – 0x002F	I&M5 – 15 [a] [a] Reserved for additional identification and maintenance data	RecordDataRead RecordDataWrite
0x0030	I&M0FilterDataSubmodul	I&M0FilterData
0x0031	I&M0FilterDataModul	I&M0FilterData
0x0032	I&M0FilterDataDevice	I&M0FilterData
0x8001	Alarm Ack High	AlarmAck-PDU
0x8002	Alarm Ack Low	AlarmAck-PDU
0x0101	ARBlockReq	IODConnectReq
0x8101	ARBlockRes	IODConnectRes
0x0102	IOCRBlockReq	IODConnectReq
0x8102	IOCRBlockRes	IODConnectRes
0x0103	AlarmCRBlockReq	IODConnectReq
0x8103	AlarmCRBlockRes	IODConnectRes
0x0104	ExpectedSubmoduleBlockReq	IODConnectReq

Value (hexadecimal)	Meaning	Use
0x8104	ModuleDiffBlock	IODConnectRes, RecordDataRead, IOCControlReq, IOSControlReq
0x0105	PrmServerBlockReq	IODConnectReq
0x8105	PrmServerBlockRes	IODConnectRes
0x0106	MCRBlockReq	IODConnectReq
0x0110	IODBlockReq, shall only be used in conjunction with connection establishment phase	IODControlReq (Prm End.req)
0x8110	IODBlockRes, shall only be used in conjunction with connection establishment phase	IODControlRes (Prm End.rsp)
0x0111	IODBlockReq, shall only be used in conjunction with a plug alarm event	IODControlReq (Prm End.req)
0x8111	IODBlockRes, shall only be used in conjunction with a plug alarm event	IODControlRes (Prm End.rsp)
0x0112	IOXBlockReq, shall only be used in conjunction with connection establishment phase	IOXControlReq (Application Ready.req)
0x8112	IOXBlockRes, shall only be used in conjunction with connection establishment phase	IOXControlRes (Application Ready.rsp)
0x0113	IOXBlockReq, shall only be used in conjunction with a plug alarm event	IOCControlReq, IOSControlReq (Application Ready.req)
0x8113	IOXBlockRes, shall only be used in conjunction with a plug alarm event	IOCControlRes IOSControlRes (Application Ready.rsp)
0x0114	ReleaseBlockReq	IODReleaseReq
0x8114	ReleaseBlockRes	IODReleaseRes
0x0115	ARRPCServerBlockReq	IODConnectReq
0x8115	ARRPCServerBlockRes	IODConnectRes
0x0116	IOXBlockReq, shall only be used in conjunction with connection establishment phase	IOXControlReq (Ready for Companion.req)
0x8116	IOXBlockRes, shall only be used in conjunction with connection establishment phase	IOXControlRes (Ready for Companion.rsp)
0x0200	PDPortDataCheck	RecordDataRead RecordDataWrite
0x0201	PDevData	RecordDataRead
0x0202	PDPortDataAdjust	RecordDataRead RecordDataWrite
0x0203	PDSyncData	RecordDataRead RecordDataWrite
0x0204	IsochronousModeData	RecordDataRead RecordDataWrite
0x0205	PDIRData	RecordDataRead RecordDataWrite
0x0206	PDIRGlobalData	RecordDataRead RecordDataWrite
0x0207	PDIRFrameData	RecordDataRead RecordDataWrite

Value (hexadecimal)	Meaning	Use
0x0209	Sub block for adjusting DomainBoundary	PDPortDataAdjust
0x020A	Sub block for checking Peers	PDPortDataCheck
0x020B	Sub block for checking LineDelay	PDPortDataCheck
0x020C	Sub block for checking MAUType	PDPortDataCheck
0x020E	Sub block for adjusting MAUType	PDPortDataAdjust
0x020F	PDPortDataReal	RecordDataRead RecordDataWrite
0x0210	Sub block for adjusting MulticastBoundary	PDPortDataAdjust
0x0211	PDInterfaceMrpDataAdjust	RecordDataRead RecordDataWrite
0x0212	PDInterfaceMrpDataReal	RecordDataRead
0x0213	PDInterfaceMrpDataCheck	RecordDataRead RecordDataWrite
0x0214	PDPortMrpDataAdjust	RecordDataRead RecordDataWrite
0x0215	PDPortMrpDataReal	RecordDataRead
0x0216	Sub block for media redundancy manager parameters	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x0217	Sub block for media redundancy client parameters	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x0218	Sub block for media redundancy RT mode for manager	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x0219	Sub block for media redundancy ring state data	PDInterfaceMrpDataReal
0x021A	Sub block for media redundancy RT ring state data	PDInterfaceMrpDataReal
0x021B	Sub block for adjusting PortState	PDPortDataAdjust
0x021C	Sub block for checking PortState	PDPortDataCheck
0x021D	Sub block for media redundancy RT mode for clients	PDInterfaceMrpDataAdjust PDInterfaceMrpDataReal
0x021E	Sub block for checking local and remote CableDelay detected by LLDP to discover a sync difference	PDPortDataCheck
0x021F	Sub block for checking local and remote MAUTypes detected by LLDP	PDPortDataCheck
0x0220	PDPortFODataReal	RecordDataRead
0x0221	Sub block for reading real fiber optic manufacturerspecific data	RecordDataRead
0x0222	PDPortFODataAdjust	RecordDataRead RecordDataWrite
0x0223	PDPortFODataCheck	RecordDataRead RecordDataWrite
0x0230	PDNCDataCheck	RecordDataRead RecordDataWrite
0x0240	PDInterfaceDataReal	RecordDataRead
0x0400	MultipleBlockHeader	RecordDataRead
0x0500	RecordDataReadQuery	RecordDataRead
0x0600	FSHelloBlock	RecordDataRead RecordDataWrite
0x0601	FSParameterBlock	RecordDataRead RecordDataWrite

Value (hexadecimal)	Meaning	Use
0x0602	FSModeBlock	RecordDataRead RecordDataWrite
0x0603 – 0x60F	Reserved for FastStartUp	RecordDataRead RecordDataWrite
0x0F00	MaintenanceItem	Alarmnotification-PDU
0x0F01	Upload selected records within Upload&RetrievalItem	Alarmnotification-PDU
0x0F02	iParameterItem	Alarmnotification-PDU
0x0F03	Retrieval selected records within Upload&RetrievalItem	Alarmnotification-PDU
0x0F04	Retrieval all stored records within Upload&RetrievalItem	Alarmnotification-PDU
other	Reserved	not used

6.2.1.2 Coding of the field BlockLength

This field shall be coded as data type Unsigned16. This field shall contain the number of octets without counting the fields Type and Length.

6.2.1.3 Coding of the field BlockVersionHigh

This field shall be coded as data type Unsigned8. The value shall be set to 0x01.

6.2.1.4 Coding of the field BlockVersionLow

This field shall be coded as data type Unsigned8. The value shall be set to 0x00 to indicate version 0. The value shall be set to 0x01 for ExpectedIdentificationData, RealIdentificationData and DiagnosisData if data for multiple APIs are conveyed.

6.2.2 Coding section related to RTA-SDU specific fields

6.2.2.1 Coding of the field AlarmType

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 230.

Table 230 – AlarmType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Diagnosis
0x0002	Process
0x0003	Pull [a]
0x0004	Plug
0x0005	Status
0x0006	Update
0x0007	Redundancy
0x0008	Controlled by supervisor
0x0009	Released
0x000A	Plug Wrong Submodule
0x000B	Return of Submodule
0x000C	Diagnosis disappears
0x000D	Multicast communication mismatch notification
0x000E	Port data change notification
0x000F	Sync data changed notification
0x0010	Isochronous mode problem notification
0x0011	Network component problem notification
0x0012	Time data changed notification
0x0013 – 0x001D	Reserved
0x001E	Upload and retrieval notification
0x001F	Pull module [b]
0x0020 – 0x007F	Manufacturer specific
0x0080 – 0x00FF	Reserved for profiles
0x0100 – 0xFFFF	Reserved

[a] With ARProperties.PullModuleAlarmAllowed(=0) subslot number 0x0001 – 0x8FFF used as “Pull submodule” and subslot number 0 used as “Pull module”.

[b] With ARProperties.PullModuleAlarmAllowed(=1) AlarmType(Pull) shall signal pulling of submodule and AlarmType(Pull module) shall signal pulling of module.

6.2.2.2 Coding of the field AlarmSpecifier

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 10: AlarmSpecifier.SequenceNumber

This field shall be incremented with every AlarmNotification. The allowed range is from 0 to 2 047. By means of the SequenceNumber the receiver detects duplications and reflects the value of the field in the AlarmAck.

Bit 11: AlarmSpecifier.ChannelDiagnosis

This field shall be coded with the values according to Table 231. For all other AlarmTypes this field shall be set to zero.

Table 231 – AlarmSpecifier.ChannelDiagnosis

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Means that the SubslotNumber contains no ChannelDiagnosis, ExtChannelDiagnosis or QualifiedChannelDiagnosis with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear)	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	Means that the SubslotNumber contains at least one ChannelDiagnosis, ExtChannelDiagnosis or QualifiedChannelDiagnosis with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear)	

Bit 12: AlarmSpecifier.ManufacturerSpecificDiagnosis
This field shall be coded with the values according to Table 232. For all other AlarmTypes this field shall be set to zero.

Table 232 – AlarmSpecifier.ManufacturerSpecificDiagnosis

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Means that the SubslotNumber contains no ManufacturerSpecificDiagnosis with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear or disappear)	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	Means that the SubslotNumber contains at least one ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance (=diagnosis) and ChannelProperties.Specifier (=appear or disappear)	

Bit 13: AlarmSpecifier.SubmoduleDiagnosisState
This field shall be coded with the values according to Table 233. For all other AlarmTypes this field shall be set to zero.

Table 233 – AlarmSpecifier.SubmoduleDiagnosisState

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Error free No DiagnosisData with ChannelProperties.Maintenance (=diagnosis) and ChannelProperties.Specifier (=appear) exists at the submodul. Furthermore, it indicates that all reported diagnosis has been cleared. An individual “disappears” notification can be omitted. However, even in this case a Manufacturer Specific Diagnosis with ChannelProperties.Maintenance (=diagnosis) and ChannelProperties.Specifier (=disappear) may be present.	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	At least one DiagnosisData with Channel-Properties.Maintenance (=diagnosis) and Channel-Properties.Specifier (=appear) exists at the submodul.	

Bit 14: AlarmSpecifier.reserved
This field shall be set to zero.

Bit 15: AlarmSpecifier.ARDiagnosisState
This field shall be coded with the values according to Table 234. For all other AlarmTypes this field shall be set to zero.

Table 234 – AlarmSpecifier.ARDiagnosisState

Value (hexadecimal)	Meaning	Usage within AlarmType
0x00	Error free No DiagnosisData with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear) exists at the AR. Furthermore, it indicates that all reported diagnosis has been cleared. An individual "disappears" notification can be omitted. However, even in this case a Manufacturer Specific Diagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=disappear) may be present.	Diagnosis, Redundancy, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification
0x01	At least one DiagnosisData with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear) exists at the AR.	

6.2.3 Coding section related to common address fields

6.2.3.1 Coding of the field API

This field shall be coded as data type Unsigned32. The default AP is addressed by 0. All other values shall not be used and are reserved for profile definitions. Each new AP shall be addressed by x with 1<= x <= 0xFFFFFFFF.

NOTE 1 The API is assigned by PROFIBUS International (PI).

NOTE 2 Depends on 6.2.4.83.8

6.2.3.2 Coding of the field SlotNumber

This field shall be coded as data type Unsigned16. The coding shall be according to Table 235.

Table 235 – SlotNumber

Value (hexadecimal)	Meaning of SlotNumber
0 – 0x7FFF	The first usable slot for modules is zero. The last usable slot for modules is 0x7FFF. It may contain gaps.
0x8000 – 0xFFFF	Reserved

6.2.3.3 Coding of the field SubslotNumber

This field shall be coded as data type Unsigned16. The coding shall be according to Table 236.

Table 236 – SubslotNumber

Value (hexadecimal)	Meaning of SubslotNumber	
0	ARProperties.PullModuleAlarmAllowed (=0)	Shall be used to code "Pull module" in conjunction with AlarmType (Pull).
	ARProperties.PullModuleAlarmAllowed (=1)	Useable subslot
0x0001 – 0x7FFF	The first usable subslot for submodules is one. The last usable subslot for submodules is 0x7FFF. There may be gaps.	
0x8000 – 0x8FFF	Used for 16 interface modules with up to 255 ports. There may be gaps.	
0x9000 – 0xFFFF	Reserved	

The subslotnumber shall be evaluated with 0x8IPP with I counting interfaces (P:=00 means interface itself) and P counting ports.

6.2.3.4 Coding of the field Index

6.2.3.4.1 General

This field shall be coded as data type Unsigned16. The range from zero to 0x7FFF shall be used to address user specific record data objects. The range from 0x8000 to 0xFFFF shall be used for protocol specific function or further protocol extensions. The range from 0x8000 to 0xFFFF is divided into four sections. It is defined for each section whether the fields ARUUID, SlotNumber and SubslotNumber shall be ignored or used. The optional field TargetARUUID shall only be present and evaluated in conjunction with the defined indices.

Furthermore, all services with write access shall only use the valid address space of the established AR context. Read services may address record data beyond this context.

6.2.3.4.2 Usage of the address parameters

The following expressions show the evaluating of the address parameters according to the index range.

Expression 1 (subslot specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

i) TargetARUUID == NIL

API, Slot Number, Subslot Number, Index shall be evaluated

ii) TargetARUUID != NIL

API, Slot Number, Subslot Number, Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

API, Slot Number, Subslot Number, Index shall be evaluated

Expression 2 (slot specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

i) TargetARUUID == NIL

API, Slot Number, Index shall be evaluated

ii) TargetARUUID != NIL

API, Slot Number, Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

API, Slot Number, Index shall be evaluated

Expression 3 (AR specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

i) TargetARUUID == NIL

not allowed

ii) TargetARUUID != NIL

Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

Index shall be evaluated

Expression 4 (API specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

API, Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

API, Index shall be evaluated

Expression 5 (device specific)

The rules to evaluate the address parameter shall be applied:

a) RPCOperationNmb == implicit Read

1) ARUUID == NIL (implicit AR)

Index shall be evaluated

2) ARUUID != NIL (explicit AR)

not allowed

b) RPCOperationNmb == explicit Read / Write

1) ARUUID == NIL (implicit AR)

not allowed

2) ARUUID != NIL (established AR)

Index shall be evaluated

6.2.3.4.3 Grouping of DiagnosisData for the diagnosis records

The diagnosis records contain different data dependent on the index. The grouping shall be according to Table 237. The identifier is used in Table 239, Table 240, Table 241, Table 242, and Table 243.

Table 237 – Grouping of DiagnosisData

Identifier	Meaning
Diagnosis in channel coding	ChannelDiagnosisData, ExtChannelDiagnosisData, and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear)
Diagnosis in all codings	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=appear)
Maintenance required in channel coding	ChannelDiagnosisData, ExtChannelDiagnosisData, and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceRequired) and ChannelProperties.Specifier(=appear)
Maintenance required in all codings	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=MaintenanceRequired) and ChannelProperties.Specifier(=appear)
Maintenance demanded in channel coding	ChannelDiagnosisData, ExtChannelDiagnosisData, and QualifiedChannelDiagnosis with ChannelProperties.Maintenance(=MaintenanceDemanded) and ChannelProperties.Specifier(=appear)
Maintenance demanded in all codings	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=MaintenanceDemanded) and ChannelProperties.Specifier(=appear)
Diagnosis, Maintenance, Qualified and Status	ChannelDiagnosisData, ExtChannelDiagnosisData, QualifiedChannelDiagnosis, ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=any) and ChannelProperties.Specifier(=appear), and ManufacturerSpecificDiagnosis with ChannelProperties.Maintenance(=diagnosis) and ChannelProperties.Specifier(=disappear)

6.2.3.4.4 Assigned numbers

The coding for user specific record data shall be according to Table 238.

Table 238 – Index (user specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0 – 0x7FFF	User specific RecordData	Expression 1 applies.

The coding for submodule specific defined record data shall be according to Table 239.

Table 239 – Index (subslot specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0x8000	ExpectedIdentificationData for one subslot	Expression 1 applies.
0x8001	RealIdentificationData for one subslot	Expression 1 applies.
0x8002 – 0x8009	Reserved	Reserved
0x800A	Diagnosis in channel coding for one subslot	Expression 1 applies.
0x800B	Diagnosis in all codings for one subslot	Expression 1 applies.
0x800C	Diagnosis, Maintenance, Qualified and Status for one subslot	Expression 1 applies.
0x800D – 0x800F	Reserved	Reserved
0x8010	Maintenance required in channel coding for one subslot	Expression 1 applies.
0x8011	Maintenance demanded in channel coding for one subslot	Expression 1 applies.
0x8012	Maintenance required in all codings for one subslot	Expression 1 applies.
0x8013	Maintenance demanded in all codings for one subslot	Expression 1 applies.
0x8014 – 0x801D	Reserved	Reserved
0x801E	SubstituteValues for one subslot	Expression 1 applies.
0x801F – 0x8027	Reserved	Reserved
0x8028	RecordInputDataObjectElement for one subslot	Expression 1 applies.
0x8029	RecordOutputDataObjectElement for one subslot	Expression 1 applies.
0x802A	PDPortDataReal for one subslot	Expression 1 applies.
0x802B	PDPortDataCheck for one subslot	Expression 1 applies.
0x802C	PDIRData for one subslot	Expression 1 applies.
0x802D	Expected PDSyncData for one subslot with SyncID value 0 for PTCPoverRTA	Expression 1 applies.
0x802E	Expected PDSyncData for one subslot with SyncID value 0 for PTCPoverRTC	Expression 1 applies.
0x802F	PDPortDataAdjust for one subslot	Expression 1 applies.
0x8030	IsochronousModeData for one subslot	Expression 1 applies.
0x8031	Expected PDSyncData for one subslot with SyncID value 1	Expression 1 applies.
0x8032 – 0x804E	Expected PDSyncData for one subslot with SyncID value 2..30	Expression 1 applies.
0x804F	Expected PDSyncData for one subslot with SyncID value 31	Expression 1 applies.
0x8050	PDInterfaceMrpDataReal for one subslot	Expression 1 applies.
0x8051	PDInterfaceMrpDataCheck for one subslot	Expression 1 applies.
0x8052	PDInterfaceMrpDataAdjust for one subslot	Expression 1 applies.
0x8053	PDPortMrpDataAdjust for one subslot	Expression 1 applies.
0x8054	PDPortMrpDataReal for one subslot	Expression 1 applies.
0x8055 – 0x805F	Reserved	Reserved
0x8060	PDPortFODataReal for one subslot	Expression 1 applies.
0x8061	PDPortFODataCheck for one subslot	Expression 1 applies.

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0x8062	PDPortFODataAdjust for one subslot	Expression 1 applies.
0x8063 – 0x806F	Reserved	Reserved
0x8070	PDNCDataCheck for one subslot	Expression 1 applies.
0x8071 – 0x807F	Reserved	Reserved
0x8080	PDInterfaceDataReal for one subslot	Expression 1 applies.
0x8081 – 0xAFEF	Reserved	Reserved
0xAFF0	I&M0	Expression 1 applies.
0xAFF1	I&M1	Expression 1 applies.
0xAFF2	I&M2	Expression 1 applies.
0xAFF3	I&M3	Expression 1 applies.
0xAFF4	I&M4	Expression 1 applies.
0xAFF5 – 0xAFFF	I&M5 – I&M15 [a] [a] Reserved for additional identification and maintenance data	Expression 1 applies.
0xB000 – 0xBFFF	Reserved for profiles	Expression 1 applies.

The coding for module specific defined record data shall be according to Table 240.

Table 240 – Index (slot specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0xC000	ExpectedIdentificationData for one slot	Expression 2 applies.
0xC001	RealIdentificationData for one slot	Expression 2 applies.
0xC002 – 0xC009	Reserved	Reserved
0xC00A	Diagnosis in channel coding for one slot	Expression 2 applies.
0xC00B	Diagnosis in all codings for one slot	Expression 2 applies.
0xC00C	Diagnosis, Maintenance, Qualified and Status for one slot	Expression 2 applies.
0xC00D – 0xC00F	Reserved	Reserved
0xC010	Maintenance required in channel coding for one slot	Expression 2 applies.
0xC011	Maintenance demanded in channel coding for one slot	Expression 2 applies.
0xC012	Maintenance required in all codings for one slot	Expression 2 applies.
0xC013	Maintenance demanded in all codings for one slot	Expression 2 applies.
0xC014 – 0xCFFF	Reserved	Reserved
0xD000 – 0xDFFF	Reserved for profiles	Expression 2 applies.

The coding for AR specific defined record data shall be according to Table 241.

Table 241 – Index (AR specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0xE000	ExpectedIdentificationData for one AR	Expression 3 applies.
0xE001	RealIdentificationData for one AR	Expression 3 applies.
0xE002	ModuleDiffBlock for one AR	Expression 3 applies.
0xE003 – 0xE009	Reserved	Reserved
0xE00A	Diagnosis in channel coding for one AR	Expression 3 applies.
0xE00B	Diagnosis in all codings for one AR	Expression 3 applies.
0xE00C	Diagnosis, Maintenance, Qualified and Status for one AR	Expression 3 applies.
0xE00D – 0xE00F	Reserved	Reserved
0xE010	Maintenance required in channel coding for one AR	Expression 3 applies.
0xE011	Maintenance demanded in channel coding for one AR	Expression 3 applies.
0xE012	Maintenance required in all codings for one AR	Expression 3 applies.
0xE013	Maintenance demanded in all codings for one AR	Expression 3 applies.
0xE014 – 0xE02F	Reserved	Reserved
0xE030	IsochronousModeData for one AR	Expression 3 applies.
0xE031 – 0xE03F	Reserved	Reserved
0xE040	MultipleWrite	Expression 3 applies.
0xE041 – 0xE04F	Reserved	Reserved
0xE050	FastStartUp data for one AR	Expression 3 applies.
0xE051 – 0xE05F	Reserved for FastStartUp	Expression 3 applies.
0xE060 – 0xEBFF	Reserved	Reserved
0xEC00 – 0xEFFF	Reserved for profiles	Expression 3 applies.

The coding for API specific defined record data shall be according to Table 242 if no AR has been established.

Table 242 – Index (API specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0xF000	RealIdentificationData for one API	Expression 4 applies.
0xF001 – 0xF009	Reserved	Reserved
0xF00A	Diagnosis in channel coding for one API	Expression 4 applies.
0xF00B	Diagnosis in all codings for one API	Expression 4 applies.
0xF00C	Diagnosis, Maintenance, Qualified and Status for one API	Expression 4 applies.
0xF00D – 0xF00F	Reserved	Reserved
0xF010	Maintenance required in channel coding for one API	Expression 4 applies.
0xF011	Maintenance demanded in channel coding for one API	Expression 4 applies.
0xF012	Maintenance required in all codings for one API	Expression 4 applies.
0xF013	Maintenance demanded in all codings for one API	Expression 4 applies.
0xF014 – 0xF01F	Reserved	Reserved
0xF020	ARData for one API	Expression 4 applies.
0xF021 – 0xF3FF	Reserved	Reserved
0xF400 – 0xF7FF	Reserved for profiles	Expression 4 applies.

The coding for device specific defined record data shall be according to Table 243.

Table 243 – Index (device specific)

Value (hexadecimal)	Meaning of index	Meaning of fields ARUUID, API, SlotNumber, SubslotNumber and TargetARUUID
0xF800 – 0xF80B	Reserved	Reserved
0xF80C	Diagnosis, Maintenance, Qualified and Status for one device	Expression 5 applies.
0xF80D – 0xF81F	Reserved	Reserved
0xF820	ARData	Expression 5 applies.
0xF821	APIData	Expression 5 applies.
0xF822 – 0xF82F	Reserved	Reserved
0xF830	LogData	Expression 5 applies.
0xF831	PDevData	Expression 5 applies.
0xF832 – 0xF83F	Reserved	Reserved
0xF840	I&M0FilterData	Expression 5 applies.
0xF841	PDRealData	Expression 5 applies.
0xF842	PDExpectedData	Expression 5 applies.
0xF843 – 0xFBFF	Reserved	Reserved
0xFC00 – 0xFFFF	Reserved for profiles	Expression 5 applies.

6.2.4 Coding section related to AL services

6.2.4.1 Coding of the field RecordDataLength

This field shall be coded as data type Unsigned32. The field RecordDataLength shall only contain the number of user octets.

6.2.4.2 Coding of the field SeqNumber

This field shall be coded as data type Unsigned16.

6.2.4.3 Coding of the field ARType

This field shall be coded as data type Unsigned16 with the values according to Table 244.

Table 244 – ARType

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	
0x0001	IOCARSingle	
0x0002	Reserved	
0x0003	IOCARCIR	For future versions
0x0004	IOCAR_IOControllerRedundant	For future versions
0x0005	IOCAR_IODeviceRedundant	For future versions
0x0006	IOSAR	The supervisor AR is a special form of the IOCARSingle
0x0007 – 0xFFFF	Reserved	

6.2.4.4 Coding of the field SessionKey

This field shall be coded as data type Unsigned16. The value of the field SessionKey shall be increased by one for each connect by the CMInitiator. The CMResponder shall use this value for each subsequent PROFINETIOServiceReq- and PROFINETIOServiceResPDU within this session.

NOTE The SessionKey allows the CMInitiator to detect sequence errors during the establishment and release phase of an AR.

This field shall contain the value 0 for the implicit AR.

6.2.4.5 Coding of the field CMInitiatorMacAdd

This field shall be coded as data type OctetString[6]. The value of the field CMInitiatorMacAdd shall be according to IEEE 802 MAC address.

6.2.4.6 Coding of the field IOCRMulticastMACAdd

This field shall be coded as data type OctetString[6]. The value of the field IOCRMulticastMACAdd shall be according to IEEE 802 MAC address.

This field shall be set according to the Table 245.

Table 245 – IOCRMulticastMACAdd

Value OUI (Multicast) (hexadecimal)	Value ExtensionIdentifier (hexadecimal)	Meaning
01-0E-CF	00-00-00 to 00-00-FF	Reserved for other applications
01-0E-CF	00-01-00	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-01-01	RT_CLASS_3 destination address
01-0E-CF	00-01-02	RT_CLASS_3 synchronization multicast address
01-0E-CF	00-01-03 to 00-01-FF	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-02-00 to 00-02-FF	RT_CLASS_2 multicast communication address
01-0E-CF	00-03-00 to 00-03-FF	Reserved for further multicast addresses within the Type 10 context
01-0E-CF	00-04-00 to FF-FF-FF	Reserved for other applications

NOTE Octet 1 contains the Individual/Group Address Bit (LSB).

The IEEE Organizationally Unique Identifier for Type 10 is 00-0E-CF.

It shall be set according to the Table 246.

Table 246 – Type 10 OUI

Value (hexadecimal)	Meaning
00-0E-CF	Global administered individual unicast
01-0E-CF	Global administered group (multicast) address
02-0E-CF	Local administered individual unicast
03-0E-CF	Local administered group (multicast) address

6.2.4.7 Coding of the field CMResponderMacAdd

This field shall be coded as data type OctetString[6]. The value of the field CMResponderMacAdd shall be according to IEEE 802 MAC address.

NOTE Octet 1 contains the Individual/Group Address Bit (lsb).

6.2.4.8 Coding of the field ARProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 2: ARProperties.State
This field shall be set according to the Table 247.

Table 247 – ARProperties.State

Value (hexadecimal)	Meaning
0x00	Backup
0x01	Primary
0x02 – 0x07	Reserved

Bit 3: ARProperties.SupervisorTakeoverAllowed
This field shall be set according to Table 248.

Table 248 – ARProperties.SupervisorTakeoverAllowed

Value (hexadecimal)	Meaning
0x00	Not allowed
0x01	Allowed

Bit 4: ARProperties.ParametrizationServer
This field shall be set according to Table 249.

Table 249 – ARProperties.ParametrizationServer

Value (hexadecimal)	Meaning
0x00	External PrmServer
0x01	CM Initiator

Bit 5 – 6: ARProperties.DataRate
This field shall be set according to Table 250.

Table 250 – ARProperties.DataRate

Value (hexadecimal)	Meaning	Usage
0x00	At least 100 Mbit/s or more	Default Shall be supported by an IO device, IO controller and IO supervisor
0x01	100 Mbit/s	May be supported by an IO device, IO controller and IO supervisor
0x02	1 Gbit/s	May be supported by an IO device, IO controller and IO supervisor
0x03	10 Gbit/s	May be supported by an IO device, IO controller and IO supervisor

Bit 7: ARProperties.reserved_1
This field shall be set to 0 and not checked by the receiver.

Bit 8: ARProperties.DeviceAccess
This field shall be set according to Table 251.

Table 251 – ARProperties.DeviceAccess

Value (hexadecimal)	Meaning
0x00	Only the submodules from the ExpectedSubmoduleBlock are accessible
0x01	Submodule access is controlled by IO device application

Bit 9-10: ARProperties.CompanionAR
This field shall be set according to Table 252.

Table 252 – ARProperties.CompanionAR

Value (hexadecimal)	Meaning	Use
0x00	Single AR	RT_CLASS_1 or RT_CLASS_2 connection establishment
0x01	First AR of a companion pair and a companion AR shall follow	RT_CLASS_3 connection establishment
0x02	Companion AR	RT_CLASS_3 connection establishment
0x03	Reserved	

Bit 11: ARProperties.AcknowledgeCompanionAR
This field shall be set according to Table 253.

Table 253 – ARProperties.AcknowledgeCompanionAR

Value (hexadecimal)	Meaning	Use
0x00	No companion AR or no acknowledge for the companion AR required	E.g. RT_CLASS_1 or RT_CLASS_2 connection establishment
0x01	Companion AR with acknowledge	RT_CLASS_3 connection establishment

Bit 12 – 23: ARProperties.reserved_2
This field shall be set according to 3.7.3.2.

Bit 24 – 30: ARProperties.reserved_3
This field shall be set to zero.

Bit 31: ARProperties.PullModuleAlarmAllowed
This field shall be set according to Table 254.

Table 254 – ARProperties.PullModuleAlarmAllowed

Value (hexadecimal)	Meaning	Use
0x00	The AlarmType(=Pull) shall signal pulling of submodule and module. The subslot number zero shall code pulling of module in conjunction with AlarmType(=Pull).	Mandatory
0x01	The AlarmType(=Pull) shall signal pulling of submodule. The AlarmType(=Pull module) shall signal pulling of module. The subslot number 0 – 0x8FFF shall code pulling of submodule in conjunction with AlarmType(=Pull).	Optional

6.2.4.9 Coding of the field IOCRProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 3: IOCRProperties.RTClass
This field shall be set according to the Table 255.

Table 255 – IOCRProperties.RTClass

Value (hexadecimal)	Meaning	Use
0x00	Reserved	
0x01	RT_CLASS_1	Data-RTC-PDU
0x02	RT_CLASS_2	Data-RTC-PDU
0x03	RT_CLASS_3	Data-RTC-PDU
0x04	RT_CLASS_UDP	UDP-RTC-PDU
0x05 – 0x07	Reserved	

Bit 4 – 10: IOCRProperties.reserved_1
This field shall be set to zero.

Bit 11: IOCRProperties.MediaRedundancy
This field shall be set according to the Table 256.

Table 256 – IOCRProperties.MediaRedundancy

Value (hexadecimal)	Meaning
0x00	No media redundant frame transfer
0x01	Media redundant frame transfer

Bit 12 – 23: IOCRProperties.reserved_2
This field shall be set according to 3.7.3.2.

Bit 24 – 31: IOCRProperties.reserved_3
This field shall be set to zero.

6.2.4.10 Coding of the field NumberOfIODataObjects

This field shall be coded as data type Unsigned16.

6.2.4.11 Coding of the field NumberOfIOCS

This field shall be coded as data type Unsigned16.

6.2.4.12 Coding of the field IOCRReference

This field shall be coded as data type Unsigned16. It is an identification tag for the CR and is used within the IOCBlockReq and IOCBlockRes to reference the DataItem. Furthermore, it is used in PDIRData of the Physical Device ASE.

6.2.4.13 Coding of the field IOCRTagHeader

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0-11: IOCRTagHeader.IOCRVLANID
This field shall be set according to the Table 257.

Table 257 – IOCRTagHeader.IOCRVLANID

Value (hexadecimal)	Meaning
0x000	No VLAN
0x001	Default VLAN
0x002 – 0xFFF	According IEEE 802.1Q

Bit 12: IOCRTagHeader.reserved
This field shall be set to zero.

Bit 13 – 15: IOCRTagHeader.IOUserPriority
This field shall be set according to the Table 258.

Table 258 – IOCRTagHeader.IOUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x05	Reserved
0x06	IO CR Priority
0x07	Reserved

6.2.4.14 Coding of the field IOCRType

This field shall be coded as data type Unsigned16 with the values according to Table 259.

Table 259 – IOCRType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Input CR
0x0002	Output CR
0x0003	Multicast Provider CR
0x0004	Multicast Consumer CR
0x0005 – 0xFFFF	Reserved

6.2.4.15 Coding of the field CMInitiatorActivityTimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 260 and Table 261.

Table 260 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess:=0

Value (decimal)	Meaning	Use
0	Reserved	
1 – 1 000	With a time base of 100 ms	The IO device monitors the time between Connect response and the subsequent first activity of the IO controller
1 001 – 65 535	Reserved	

Table 261 – CMInitiatorActivityTimeoutFactor with ARProperties.DeviceAccess:=1

Value (decimal)	Meaning	Use
0 – 99	Reserved	
100 – 1 000	With a time base of 100 ms	The IO device monitors the time between Connect response and the subsequent Read and Write record activity of the IO controller
1 001 – 65 535	Reserved	

6.2.4.16 Coding of the field StationNameLength

This field shall be coded as data type Unsigned16.

6.2.4.17 Coding of the field CMInitiatorStationName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.32.1.

NOTE The field CMInitiatorStationName is not terminated by zero.

6.2.4.18 Coding of the field ParameterServerStationName

This field shall be coded as in 6.2.4.17.

6.2.4.19 Coding of the field ProviderStationName

This field shall be coded as in 6.2.4.17.

6.2.4.20 Coding of the field IODataObjectFrameOffset

This field shall be coded as data type Unsigned16. It is used within the IOCRBlockReq to reference the offset of the DataItem. It shall be in a range off 0 to 1 439. The maximum number of used octets shall not exceed the maximum C_SDU size.

6.2.4.21 Coding of the field IOCSFrameOffset

This field shall be coded as data type Unsigned16. It is used within the IOCRBlockReq to reference the offset of the IOCS. It shall be in a range off 0 to 1 439. The maximum number of used octets shall not exceed the maximum C_SDU size.

6.2.4.22 Coding of the field LengthIOCS

This field shall be coded as data type Unsigned8 and shall be set according to the Table 262.

Table 262 – LengthIOCS

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	One octet for IOCS
0x02 – 0xFF	Reserved

6.2.4.23 Coding of the field LengthIOPS

This field shall be coded as data type Unsigned8 and shall be set according to the Table 263.

Table 263 – LengthIOPS

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	One octet for IOPS
0x02 – 0xFF	Reserved

6.2.4.24 Coding of the field LengthData

This field shall be coded as data type Unsigned16. The values shall be in the range from 0 to 1 439.

6.2.4.25 Coding of the field NumberOfAPIs

This field shall be coded as data type Unsigned16.

NOTE The value 0 is only used in conjunction with ARProperties.DeviceAccess =1.

6.2.4.26 Coding of the field AlarmCRProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0: AlarmCRProperties.Priority
This field shall be set according to the Table 264.

Table 264 – AlarmCRProperties.Priority

Value (hexadecimal)	Meaning
0x00	User priority (default), the priority given by the user is used and two alarm resources are available
0x01	Use only low priority, user priority is ignored and only one alarm resource is available

Bit 1: AlarmCRProperties.Transport
This field shall be set according to Table 265.

Table 265 – AlarmCRProperties.Transport

Value (hexadecimal)	Meaning	Usage
0x00	RTA_CLASS_1	Alarm CR uses DATA-RTA-PDU
0x01	RTA_CLASS_UDP	Alarm CR uses UDP-RTA-PDU

Bit 2 – 23: AlarmCRProperties.reserved_1
This field shall be set according to 3.7.3.2.

Bit 24 – 31: AlarmCRProperties.reserved_2
This field shall be set to zero.

6.2.4.27 Coding of the field AlarmCRTagHeaderHigh

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0-11: AlarmCRTagHeaderHigh.AlarmCRVLANID
This field shall be set according to the Table 266.

Table 266 – AlarmCRTagHeaderHigh.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	No VLAN
0x001	Default VLAN
0x002 – 0xFFF	According IEEE 802.1Q

Bit 12: AlarmCRTagHeaderHigh.reserved
This field shall be set to zero.

Bit 13 – 15: AlarmCRTagHeaderHigh.AlarmUserPriority
This field shall be set according to the Table 267.

Table 267 – AlarmCRTagHeaderHigh.AlarmUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x05	Reserved
0x06	Alarm CR Priority High
0x07	Reserved

6.2.4.28 Coding of the field AlarmCRTagHeaderLow

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0-11: AlarmCRTagHeaderLow.AlarmCRVLANID
This field shall be set according to the Table 268.

Table 268 – AlarmCRTagHeaderLow.AlarmCRVLANID

Value (hexadecimal)	Meaning
0x000	No VLAN
0x001	Default VLAN
0x002 – 0xFFF	According IEEE 802.1Q

Bit 12: AlarmCRTagHeaderLow.reserved
This field shall be set to zero.

Bit 13 – 15: AlarmCRTagHeaderLow.AlarmUserPriority
This field shall be set according to the Table 269.

Table 269 – AlarmCRTagHeaderLow.AlarmUserPriority

Value (hexadecimal)	Meaning
0x00 – 0x04	Reserved
0x05	Alarm CR Priority Low
0x06 – 0x07	Reserved

6.2.4.29 Coding of the field AlarmSequenceNumber

This field shall be coded as data type Unsigned16 with the values according to Table 270.

Table 270 – AlarmSequenceNumber

Value (hexadecimal)	Meaning	Use
0x000 – 0x7FF	Mirrors the sequence number of the plug alarm notification	It provides the relation between Plug Alarm notification and the ControlBlockPlug within Application Ready
0x800 – 0xFFFF	Reserved	

6.2.4.30 Coding of the field AlarmCRType

This field shall be coded as data type Unsigned16 with the values according to Table 271.

Table 271 – AlarmCRType

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001	Alarm CR
0x0002 – 0xFFFF	Reserved

6.2.4.31 Coding of the field RTATimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 272. The time base is 100 ms. The RTATimeout is calculated:

$$RTATimeout = RTATimeoutFactor \times 100 \text{ ms} \tag{48}$$

Table 272 – RTATimeoutFactor

Value (hexadecimal)	Meaning	Use
0x0000	Reserved	
0x0001 – 0x0064	Mandatory	DATA-RTA-PDU or UDP-RTA-PDU acknowledge monitoring
0x0065 – 0xFFFF	Optional	DATA-RTA-PDU or UDP-RTA-PDU acknowledge monitoring

NOTE The value of the RTATimeoutFactor depends on the conveyance time of the DATA-RTA-PDU or UDP-RTA-PDU between two peers.

6.2.4.32 Coding of the field RTARetries

This field shall be coded as data type Unsigned16 with the values according to Table 273.

Table 273 – RTARetries

Value (hexadecimal)	Meaning
0x0000 – 0x0002	Reserved
0x0003 – 0x000F	Mandatory
0x0010 – 0xFFFF	Reserved

6.2.4.33 Coding of the field PROFINETIOConstantValue

This field shall be coded as data type structure containing the following elements:

- Data1 as Unsigned32 (0xDEA00000)
- Data2 as Unsigned16 (0x6C97)
- Data3 as Unsigned16 (0x11D1)
- Data4 as array of two Unsigned8, Octet 1 (0x82) to Octet 2 (0x71)

NOTE The ordering of transmission of the Unsigned32 or Unsigned16 values is according to the RPC Flag DRep (Little Endian or Big Endian) if it is part of the RPCHeader or NDREPMapPDU. If it is part of the NDRDataxxx PDUs then only Big Endian Format is used.

The value of the field PROFINETIOConstantValue shall be DEA00000-6C97-11D1-8271.

6.2.4.34 Coding of the field AddressResolutionProperties

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 2: AddressResolutionProperties.Protocol
This field shall be set according to Table 274.

Table 274 – AddressResolutionProperties.Protocol

Value (hexadecimal)	Meaning	Use
0x00	Reserved	
0x01	DNS	Multicast consumer uses DNS to resolve the multicast provider source MAC address.
0x02	DCP	Multicast consumer uses DCP to resolve the multicast provider source MAC address.
0x03 – 0x07	Reserved	

Bit 3 – 7: AddressResolutionProperties.reserved_1
This field shall be set according to 3.7.3.2.

Bit 8 – 15: AddressResolutionProperties.reserved_2
This field shall be set to zero.

Bit 16 – 31: AddressResolutionProperties.Factor
This field shall be coded with the values according to Table 275. The time base is one second. The AddressResolutionInterval is calculated:

$$\text{AddressResolutionInterval} = \text{AddressResolutionProperties.Factor} \times 1\ \text{s} \quad (49)$$

Table 275 – AddressResolutionProperties.Factor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0064	Mandatory
0x0065 – 0xFFFF	Optional

6.2.4.35 Coding of the field MCITimeoutFactor

This field shall be coded as data type Unsigned16 with the values according to Table 276. The time base shall be 100 ms. The MCIMonitoringInterval is calculated:

$$\text{MCIMonitoringInterval} = \text{MCITimeoutFactor} \times 100\ \text{ms} \quad (50)$$

Table 276 – MCITimeoutFactor

Value (hexadecimal)	Meaning
0x0000 – 0x0064	Mandatory
0x0065 – 0xFFFF	Reserved

6.2.4.36 Coding of fields related to Instance, DeviceID, VendorID

6.2.4.36.1 General

The Unsigned16 fields Instance, DeviceID, and VendorID are divided into two Unsigned8 fields to define the order of the bytes inside the ObjectUUID OctetString.

6.2.4.36.2 Coding of the field InstanceLow

This field shall be coded as data type Unsigned8.

6.2.4.36.3 Coding of the field InstanceHigh

This field shall be coded as data type Unsigned8.

NOTE The value 0 for both InstanceHigh and InstanceLow is recommended for single instance devices.

6.2.4.36.4 Coding of the field DeviceIDLow

This field shall be coded as data type Unsigned8.

6.2.4.36.5 Coding of the field DeviceIDHigh

This field shall be coded as data type Unsigned8.

NOTE The value 0 for both DeviceIDHigh and DeviceIDLow is reserved.

6.2.4.36.6 Coding of the field VendorIDLow

This field shall be coded as data type Unsigned8.

6.2.4.36.7 Coding of the field VendorIDHigh

This field shall be coded as data type Unsigned8. The value 0xFF is reserved for profiles.

NOTE The value 0 for both VendorIDHigh and VendorIDLow is reserved for IEC 61158–6–3 devices.

6.2.4.37 Coding of the field ModuleIdentNumber

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 277.

Table 277 – ModuleIdentNumber

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0xFFFFFFFF	Manufacturer specific

6.2.4.38 Coding of the field SubmoduleIdentNumber

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 278.

Table 278 – SubmoduleIdentNumber

Value (hexadecimal)	Meaning
0x00000000	Assigned to the subslot zero only, and Manufacturer specific for subslots > 0
0x00000001 – 0xFFFFFFFF	Manufacturer specific

The numbering for device identification is hierarchical. The toplevel is the DeviceIdentNumber which contains an administrative uniquely assigned number. The second level is the ModuleIdentNumber that is unique in the scope of the DeviceIdentNumber. The third level is the SubmoduleIdentNumber that is unique in the scope of the ModuleIdentNumber.

6.2.4.39 Coding of the field NumberOfARs

This field shall be coded as data type Unsigned16.

6.2.4.40 Coding of the field NumberOfIOCRs

This field shall be coded as data type Unsigned16.

6.2.4.41 Coding of the field SubmoduleDataLength

This field shall be coded as data type Unsigned16. The range shall be between 0 and 1 439.

6.2.4.42 Coding of the field NumberOfModules

This field shall be coded as data type Unsigned16.

6.2.4.43 Coding of the field NumberOfSlots

This field shall be coded as data type Unsigned16.

6.2.4.44 Coding of the field NumberOfSubmodules

This field shall be coded as data type Unsigned16.

6.2.4.45 Coding of the field NumberOfSubslots

This field shall be coded as data type Unsigned16.

6.2.4.46 Coding of the field ARUUID

This field shall be coded as data type UUID. The value NIL indicates the usage of the implicit AR. All other values identify established ARs.

6.2.4.47 Coding of the field TargetARUUID

This field shall be coded as data type UUID. The value NIL indicates the usage of the implicit AR. All other values identify established ARs.

6.2.4.48 Coding of the field ActualLocalTimeStamp

This field shall be coded as data type Unsigned64. The value contains the current cycle count.

6.2.4.49 Coding of the field LocalTimeStamp

This field shall be coded as data type Unsigned64. The value contains the cycle count.

6.2.4.50 Coding of the field NumberOfLogEntries

This field shall be coded as data type Unsigned16. The value contains the number of log entries. The minimum number shall be 16 for an IO device and 4Kbyte an IO controller.

6.2.4.51 Coding of the field EntryDetail

This field shall be coded as data type Unsigned32 according to protocol machine behavior. The value zero shall be used for no detail.

6.2.4.52 Coding of the field AdditionalValue1 and AdditionalValue2

This field shall be coded as data type Unsigned16. The values shall contain additional user information within negative responses. The value zero indicates no further information.

6.2.4.53 Coding of the field ControlBlockProperties

The coding of this field shall be according to 3.7.3.4, Table 279 and Table 280.

Table 279 – ControlBlockProperties in conjunction with ControlCommand.ApplicationReady

Bitposition	Value (hexadecimal)	Meaning
Bit 0	0x00	Wait for explicit ControlCommand.ReadyForCompanion
	0x01	Implicit ControlCommand.ReadyForCompanion
Bit 1 – 15		Shall be set according to 3.7.3.2.

Table 280 – ControlBlockProperties in conjunction with the other values of the field ControlCommand

Bitposition	Value (hexadecimal)	Meaning
Bit 0 – 15		Shall be set according to 3.7.3.2.

6.2.4.54 Coding of the field ControlCommand

The coding of this field shall be according to 3.7.3.4. Only one of the following bits shall be set. The individual bits shall have the following meaning:

Bit 0: ControlCommand.PrmEnd
This field shall be set according to the Table 281.

Table 281 – ControlCommand.PrmEnd

Value (hexadecimal)	Meaning	Use
0x00	no PrmEnd	
0x01	The IO controller has finished the transmission of the stored start-up parameter.	IODControlReq

Bit 1: ControlCommand.ApplicationReady
This field shall be set according to the Table 282.

Table 282 – ControlCommand.ApplicationReady

Value (hexadecimal)	Meaning	Use
0x00	No Application Ready	
0x01	The IO device has finished the start-up of its application or a new module or submodule was plugged and is ready to operate.	IOCControlReq, IOSControlReq

Bit 2: ControlCommand.Release
This field shall be set according to the Table 283.

Table 283 – ControlCommand.Release

Value (hexadecimal)	Meaning	Use
0x00	No Release	
0x01	The IO controller terminates the AR.	IODReleaseReq

Bit 3: ControlCommand.Done
This field shall be set according to the Table 284.

Table 284 – ControlCommand.Done

Value (hexadecimal)	Meaning	Use
0x00	No Done	
0x01	Acknowledge is sent.	IODReleaseRes, IOXControlRes, IODControlRes

Bit 4: ControlCommand.ReadyForCompanion
This field shall be set according to the Table 285.

Table 285 – ControlCommand.ReadyForCompanion

Value (hexadecimal)	Meaning	Use
0x00	Not ready for companion	
0x01	The IO device has finished the start-up of its application and is ready for the companion AR.	IOCControlReq, IOSControlReq

Bit 4 to 15: ControlCommand.reserved
This field shall be set to zero.

6.2.4.55 Coding of the field DataDescription

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: DataDescription.Type
This field shall be set according to the Table 286.

Table 286 – DataDescription.Type

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Input
0x02	Output
0x03	Reserved

Bit 2-15: DataDescription.reserved
This field shall be set to zero.

6.2.4.56 Coding of the field DataLength

This field shall be coded as data type Unsigned16. The values shall contain the length of the C_SDU. The range is from 40 to 1 440 for RT_CLASS_1 and RT_CLASS_2. The range for RT_CLASS_3 is from 0 to 1 440. The range for RT_CLASS_UDP is from 12 to 1 440.

Note The field DataLength is in minimum the sum of all DataItems. It can also be larger.

6.2.4.57 Coding of the field GAP

This field shall be coded as data type Unsigned8. The values shall be set to zero.

6.2.4.58 Coding of the field Padding

This field shall be coded as data type Unsigned8. The values shall be set to zero.

6.2.4.59 Coding of the field RTCPadding

This field shall be coded as data type Unsigned8.

6.2.4.60 Coding of the field RTAPadding

This field shall be coded as data type Unsigned8.

6.2.4.61 Coding of the field RWPadding

This field shall be coded as data type Unsigned8. The values shall be set to zero.

6.2.4.62 Coding of the field SendClockFactor

These fields shall be coded as data type Unsigned16. The time base is 31,25 µs. The value range is from 1 to 128. It is mandatory to support the value 32.

NOTE The SendClockTime corresponds with a value range from 31,25 µs to 4 000 µs. A maximum Ethernet frame contains 1 518 or 1 522 Bytes. To transfer this kind of frame at a data rate of 100 Mbit/s the SendClockTime should be greater than 122,24 µs and at a data rate of 1 Gbit/s the SendClockTime should be greater than 12,224 µs.

Used within the PDSyncData block, this field shall only be valid if the field SyncFrameAddress.MulticastSelection contains the value 0.

6.2.4.63 Coding of the field ReductionRatio

These fields shall be coded as data type Unsigned16 according to Table 287. The value range is from 1 to 16 384 in discrete steps.

Table 287 – Values of ReductionRatio

Value (decimal)	Meaning		Use
1	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
2	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
4	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
8	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
16	Mandatory	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
32	Mandatory, optional for RT_CLASS_3	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
64	Mandatory, optional for RT_CLASS_3	RT_CLASS_x, optional for RT_CLASS_UDP	With all SendClockFactor values
128	Mandatory, optional for RT_CLASS_3	RT_CLASS_x, RT_CLASS_UDP	With all SendClockFactor values
256	Mandatory, optional for RT_CLASS_3	RT_CLASS_x, RT_CLASS_UDP	For RT_CLASS_UDP with all SendClockFactor values, for RT_CLASS_x with SendClockFactor up to 64
512	Mandatory, optional for RT_CLASS_3	RT_CLASS_x, RT_CLASS_UDP	For RT_CLASS_UDP with all SendClockFactor values, for RT_CLASS_x with SendClockFactor up to 32
1 024	Mandatory	RT_CLASS_UDP	With all SendClockFactor values
2 048	Mandatory	RT_CLASS_UDP	With all SendClockFactor values
4 096	Mandatory	RT_CLASS_UDP	With all SendClockFactor values
8 192	Mandatory	RT_CLASS_UDP	Only with SendClockFactor values less or equal 64
16 384	Mandatory	RT_CLASS_UDP	Only with SendClockFactor values less or equal 32
16 385 – 65 535	Reserved		
other	Optional		

6.2.4.64 Coding of the field Phase

These fields shall be coded as data type Unsigned16 according to Table 288. It shall be less or equal to the related ReductionRatio.

Table 288 – Values of Phase

Value (decimal)	Meaning	Use
0	Reserved	
1 – 16 384	Mandatory	but shall not exceed the value of ReductionRatio
16 385 – 65 535	Reserved	

6.2.4.65 Coding of the field Sequence

These fields shall be coded as data type Unsigned16 according to Table 289.

Table 289 – Values of Sequence

Value (hexadecimal)	Meaning	Use
0	Mandatory, sequence undefined	Provider protocol machines defines the sequence for each phase
0x0001 – 0xFFFF	Optional, sequence defined	Provider protocol machines uses the given sequence for each phase

6.2.4.66 Coding of the field DataHoldFactor and WatchdogFactor

These fields shall be coded as data type Unsigned16. The time base is SendClockFactor multiplied with ReductionRatio of the monitored consumer. The WatchdogTime and the DataHoldTime are calculated:

$$\text{WatchdogTime} = \text{WatchdogFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31{,}25\ \mu\text{s} \qquad (51)$$

$$\text{DataHoldTime} = \text{DataHoldFactor} \times \text{SendClockFactor} \times \text{ReductionRatio} \times 31{,}25\ \mu\text{s} \qquad (52)$$

The value range for Data-RTC-PDUs is from 0x0003 to 0x1E00. The DataHoldTime and WatchdogTime shall equal or less 1,92 s.

The value range for UDP-RTC-PDUs is from 0x0003 to 0xF000. The DataHoldTime and WatchdogTime shall equal or less 61,44 s.

The usage of the DataHoldFactor is defined in Table 290.

Table 290 – DataHoldFactor

Value (hexadecimal)	Meaning	Description
0x0000	Reserved	
0x0001 – 0x0002	Optional	An expiration of the time leads to an AR termination.
0x0003 – 0x00FF	Mandatory	An expiration of the time leads to an AR termination.
0x0100 – 0x1E00	Optional	An expiration of the time leads to an AR termination.
0x1E01 – 0xFFFF	Reserved	

The usage of the WatchdogFactor is defined in Table 291.

Table 291 – WatchdogFactor

Value (hexadecimal)	Meaning	Description
0x0000	Reserved	
0x0001 – 0x0002	Optional	A value “1” leads to an AR termination for one missed Frame.
0x0003 – 0x00FF	Mandatory	An expiration of the time leads to a check of the corresponding DataHoldFactor.
0x0100 – 0x1E00	Optional	An expiration of the time leads to a check of the corresponding DataHoldFactor.
0x1E01 – 0xFFFF	Reserved	

6.2.4.67 Coding of the field FrameSendOffset

These fields shall be coded as data type Unsigned32. The time base is 1 ns. The value of the field FrameSendOffset shall be set according to Equation (53).

$$\text{FrameSendOffset} < \text{SendClockFactor} \times 31{,}25\ \mu\text{s} \qquad (53)$$

The usage is defined in Table 292.

Table 292 – Values of FrameSendOffset

Value (decimal)	Meaning	Use
0 – 0x003D08FF	Optional, relative send offset to the start of the related reduction ratio and phase	Optimized scheduling
0x003D0900 – 0xFFFFFFFE	Reserved	Not used
0xFFFFFFFF	Mandatory, best effort	Provider protocol machines sends the frame as soon as possible

6.2.4.68 Coding section related to PNIOStatus

6.2.4.68.1 General

In general, the value ErrorCode=0, ErrorDecode=0, ErrorCode1=0, and ErrorCode2=0 shall be used to indicate “okay”.

Furthermore, in case of an illegal combination of address parameters within an IODReadReq the value ErrorCode=“IODReadRes”, ErrorDecode=“PNIORW”, ErrorCode1=“access-invalid area” and ErrorCode2 may used to indicate the faulty parameter.

NOTE An illegal address combination is for example TargetARUUID=NULL and ARUUID=NULL in ReadExpectedIdentification service.

6.2.4.68.2 Coding of the field ErrorCode

This field shall be coded as data type Unsigned8. The usage is defined in Table 293.

Table 293 – Values of ErrorCode for negative responses

Value (hexadecimal)	Meaning	Use
0 – 0x80	Reserved	
0x81	PNIO	LogData
0x82 – 0xCE	Reserved	
0xCF	RTA error	Within the ERR-RTA-PDU and UDP-RTA-PDU
0xD0 – 0xD9	Reserved	
0xDA	AlarmAck	Within the DATA-RTA-PDU and UDP-RTA-PDU
0xDB	IODConnectRes	Within the CL-RPC-PDU
0xDC	IODReleaseRes	Within the CL-RPC-PDU
0xDD	IODControlRes, IOCControlRes, IOSControlRes	Within the CL-RPC-PDU
0xDE	IODReadRes	Within the CL-RPC-PDU only used with ErrorDecode=PNIORW
0xDF	IODWriteRes	Within the CL-RPC-PDU only used with ErrorDecode=PNIORW
0xE0 – 0xEF	Reserved	
0xF0 – 0xFF	Reserved	

6.2.4.68.3 Coding of the field ErrorDecode

This field shall be coded as data type Unsigned8. The usage is defined in Table 294.

Table 294 – Values of ErrorDecode

Value (decimal)	Meaning	Use
0x00 – 0x7F	Reserved	
0x80	PNIORW [a]	Used in context with user error codes of the services Read, Write, Read Input Data, Read Output Data, Read Device Diagnosis, Read AR Data, Read Logbook, Read Expected Identification, Read Real Identification
0x81	PNIO	Used in context with other services or internal e.g. RPC errors
0x82 – 0xFF	Reserved	

[a] Equivalent to IEC 61158–6–3 (DPV1)

6.2.4.68.4 Coding of the field ErrorCode1 and ErrorCode2

These fields shall be coded as data type Unsigned8.

The field ErrorDecode defines the coding and meaning of ErrorCode1 and ErrorCode2.

The ErrorDecode value PNIORW indicate that the parameters ErrorCode1 shall be set according Table 295. The ErrorCode1 consists of ErrorClass and ErrorDecode.

Table 295 – Coding of ErrorCode1 with ErrorDecode PNIORW

ErrorCode1		
ErrorClass (decimal) **Bit7 – 4**	**Meaning**	**ErrorCode (decimal)** **Bit3 – 0**
0 to 9	not specified	not specified
10	application	0 = read error 1 = write error 2 = module failure 3 = not specified 4 = not specified 5 = not specified 6 = not specified 7 = busy 8 = version conflict 9 = feature not supported 10 = User specific 1 11 = User specific 2 12 = User specific 3 13 = User specific 4 14 = User specific 5 15 = User specific 6
11	Access	0 = invalid index 1 = write length error 2 = invalid slot/subslot 3 = type conflict 4 = invalid area 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10= backup 11 = User specific 7 12 = User specific 8 13 = User specific 9 14 = User specific 10 15 = User specific 11
12	resource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4 = not specified 5 = not specified 6 = not specified 7 = not specified 8 = User specific 12 9 = User specific 13 10 = User specific 14 11 = User specific 15 12 = User specific 16 13 = User specific 17 14 = User specific 18 15 = User specific 19
13 to 15	User specific	
NOTE Not specified values are used to serve legacy codes and are intended to be passed unchanged to the application.		

The ErrorDecode value PNIORW indicates that the parameter ErrorCode2 shall be set user specific.

The ErrorDecode value PNIO indicates that the parameter ErrorCode1 shall be set according Table 296.

Table 296 – Values of ErrorCode1 and ErrorCode2 for ErrorDecode with the value PNIO

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
0	Reserved	0 – 255	Reserved
1	Connect Parameter Error Faulty ARBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		13	Error in Parameter NameOfStation
		14 – 255	Reserved
2	Connect Parameter Error Faulty IOCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		23	Error in Parameter IOCSFrameOffset production
		24 – 255	Reserved
3	Connect Parameter Error Faulty ExpectedSubmoduleBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		7	Error in Parameter SubmoduleLength production
		8 – 255	Reserved
4	Connect Parameter Error Faulty AlarmCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		15	Error in Parameter AlarmCRTagHeaderLow
		16 – 255	Reserved
5	Connect Parameter Error Faulty PrmServerBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		8	Error in Parameter ParameterServerStationName
		9 – 255	Reserved
6	Connect Parameter Error Faulty MCRBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		8	Error in Parameter ProviderStationName
		9 – 255	Reserved
7	Connect Parameter Error Faulty ARRPCBlockReq	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		4	Error in Parameter InitiatorRPCServerPort
		5 – 255	Reserved

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
8	Read Write Record Parameter Error Faulty Record	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		12	Error in Parameter TargetARUUID
		13 – 255	Reserved
9 – 19	Reserved	0 – 255	Reserved
20	IODControl Parameter Error Faulty ControlBlockConnect	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
21	IODControl Parameter Error Faulty ControlBlockPlug	0	Error in Parameter BlockType
		…	…
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
22	IOXControl Parameter Error Faulty ControlBlock after a connection establishment	0	Error in Parameter BlockType
		…	…
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
23	IOXControl Parameter Error Faulty ControlBlock after a plug alarm	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
24 – 39	Reserved	0 – 255	Reserved
40	Release Parameter Error Faulty ReleaseBlock	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		7	Error in Parameter ControlBlockProperties
		8 – 255	Reserved
41 – 49	Reserved	0 – 255	Reserved
50	Response Parameter Error Faulty ARBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		8	Error in Parameter ResponderUDPRTPort
		9 – 255	Reserved

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
51	Response Parameter Error Faulty IOCRBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		6	Error in Parameter FrameID
		7 – 255	Reserved
52	Response Parameter Error Faulty AlarmCRBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		6	Error in Parameter MaxAlarmDataLengt
		7 – 255	Reserved
53	Response Parameter Error Faulty ModuleDiffBlock	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		13	Error in Parameter SubmoduleState
		14 – 255	Reserved
54	Response Parameter Error Faulty ARRPCBlockRes	0	Error in Parameter BlockType
		1	Error in Parameter Length
		…	…
		4	Error in Parameter ResponderRPCServerPort
		5 – 255	Reserved
55 – 59	Reserved	0 – 255	Reserved
60	AlarmAck Error Codes	0	Alarm Type Not Supported
		1	Wrong Submodule State
		2 – 255	Reserved
61	CMDEV	0	State conflict
		1	Resource
		2 – 255	Usage see protocol machines and stored as Logbook entries
62	CMCTL	0	State conflict
		1	Timeout
		2	No data send
		3 – 255	Reserved
63	NRPM	0	No DCP active
		1	DNS Unknown_RealStationName
		2	DCP No_RealStationName
		3	DCP Multiple_RealStationName
		4	DCP No_StationName
		5	No_IP_Addr
		6	DCP_Set_Error
		7 – 255	Reserved

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
64	RMPM	0	ArgsLength invalid
		1	Unknown Blocks
		2	IOCR Missing
		3	Wrong AlarmCRBlock count
		4	Out of AR Resources
		5	AR UUID unknown
		6	State conflict
		7	Out of Provider, Consumer, or Alarm Resources
		8	Out of Memory
		9 – 255	Reserved
65	ALPMI	0	Invalid state
		1	Wrong ACK-PDU
		2 – 255	Reserved
66	ALPMR	0	Invalid state
		1	Wrong Notification PDU
		2 – 255	Reserved
67	LMPM	0 – 255	Usage see protocol machines and stored as Logbook entries
68	MMAC	0 – 255	Usage see protocol machines and stored as Logbook entries
69	RPC	0 – 255	See Table 297
70	APMR	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
71	APMS	0	Invalid state
		1	LMPM signaled an error
		2	Timeout
		3 – 255	Reserved
72	CPM	0	Invalid state
		1 – 255	Reserved
73	PPM	0	Invalid state
		1 – 255	Reserved
74	Used by DCPUCS	0	Invalid state
		1	LMPM signaled an error
		2	Timeout
		3 – 255	Reserved
75	Used by DCPUCR	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
76	Used by DCPMCS	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
77	Used by DCPMCR	0	Invalid state
		1	LMPM signaled an error
		2 – 255	Reserved
78	FSPM	0 – 255	Usage see protocol machines and stored as Logbook entries
79 – 252	Reserved	0 – 255	Reserved
253	Used by RTA for protocol error (RTA_ERR_CLS_PROTOCOL)	0	Reserved
		1	Error within the coordination of sequence numbers (RTA_ERR_CODE_SEQ) error
		2	Instance closed (RTA_ERR_ABORT)
		3	AR out of memory (RTA_ERR_ABORT)
		4	AR add provider or consumer failed (RTA_ERR_ABORT)
		5	AR consumer missing (RTA_ERR_ABORT)
		6	AR cmi timeout (RTA_ERR_ABORT)
		7	AR alarm-open failed (RTA_ERR_ABORT)
		8	AR alarm-send.cnf(-) (RTA_ERR_ABORT)
		9	AR alarm-ack- send.cnf(-) (RTA_ERR_ABORT)
		10	AR alarm data too long (RTA_ERR_ABORT)
		11	AR alarm.ind(err) (RTA_ERR_ABORT)
		12	AR rpc-client call.cnf(-) (RTA_ERR_ABORT)
		13	AR abort.req (RTA_ERR_ABORT)
		14	AR re-run aborts existing (RTA_ERR_ABORT)
		15	AR release.ind received (RTA_ERR_ABORT)
		16	AR device deactivated (RTA_ERR_ABORT)
		17	AR removed (RTA_ERR_ABORT)
		18	AR protocol violation (RTA_ERR_ABORT)
		19	AR name resolution error (RTA_ERR_ABORT)
		20	AR RPC-Bind error (RTA_ERR_ABORT)

Value ErrorCode1 (decimal)	Meaning	Value ErrorCode2 (decimal)	Meaning/Usage
		21	AR RPC-Connect error (RTA_ERR_ABORT)
		22	AR RPC-Read error (RTA_ERR_ABORT)
		23	AR RPC-Write error (RTA_ERR_ABORT)
		24	AR RPC-Control error (RTA_ERR_ABORT)
		25	AR forbidden pull or plug after check.rsp and before in- data.ind (RTA_ERR_ABORT)
		26	AR AP removed (RTA_ERR_ABORT)
		27	AR link down (RTA_ERR_ABORT)
		28	AR could not register multicast-mac address (RTA_ERR_ABORT)
		29	Not synchronized (cannot start companion-ar) (RTA_ERR_ABORT)
		30	Wrong topology (cannot start companion-ar) (RTA_ERR_ABORT)
		31	DCP, station-name changed (RTA_ERR_ABORT)
		32	DCP, reset to factory-settings (RTA_ERR_ABORT)
		33	Can't start companion-AR because a 0x8ipp submodule in the first AR... (RTA_ERR_ABORT)
		34	No irdata record yet (RTA_ERR_ABORT)
		35	PDEV (RTA_ERROR_ABORT)
		36 – 200	Reserved
		201 – 255	Manufacturer specific
254	Reserved	0 – 255	Reserved
255	User specific	0 – 255	User specific

Table 297 – Values of ErrorCode2 for ErrorCode1 = RPC

Value (decimal)	Definition	Meaning
0	Reserved	
1	CLRPC_ERR_REJECTED	Endpoint mapper or server did reject the call. For further details see Table 157 and Table 158
2	CLRPC_ERR_FAULTED	Server had a fault while executing the call. For further details see Table 157 and Table 158
3	CLRPC_ERR_TIMEOUT	Endpoint mapper or server did not respond
4	CLRPC_ERR_IN_ARGS	Broadcast or maybe “ndr_data” too large
5	CLRPC_ERR_OUT_ARGS	Server sent back more than “alloc_len”
6	CLRPC_ERR_DECODE	Result of endpoint mapper “lookup” could not be decoded
7	CLRPC_ERR_PNIO_OUT_ARGS	Out-args not “PN IO signature”, too short or inconsistent
8	CLRPC_ERR_PNIO_APP_TIMEOUT	RPC call was terminated after RPC application timeout
9 to 255	Reserved	

6.2.4.69 Coding of the field ModuleState

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 298.

Table 298 – ModuleState

Value (hexadecimal)	Meaning	Use
0x0000	No module	E.g.module not plugged
0x0001	Wrong module	E.g. ModuleIdentNumber wrong
0x0002	Proper module	Module is okay but at least one submodule is locked, wrong or missing
0x0003	Substitute	Module is not the same as requested – but compatible. The IO device was able to adapt by its own decision
0x0004 – 0xFFFF	Reserved	

6.2.4.70 Coding of the field SubmoduleState

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

6.2.4.70.1 Coding if SubmoduleState.FormatIndicator == 1

Bit 0 – 2: SubmoduleState.AddInfo
This field shall be set according to the Table 299.

Table 299 – SubmoduleState.AddInfo

Value (hexadecimal)	Meaning	Description
0x00	None	
0x01	Takeover is not allowed	This Submodul is not available for takeover by IOSAR
0x02 – 0x07	Reserved	

Bit 3: SubmoduleState.QualifiedInfo
This field shall be set according to the Table 300.

Table 300 – SubmoduleState.QualifiedInfo

Value (hexadecimal)	Meaning	Description
0x00	No QualifiedInfo available	No Channel of the Submodule contains QualifiedChannelDiagnosis
0x01	QualifiedInfo available	At least one Channel of the Submodule contains QualifiedChannelDiagnosis

Bit 4: SubmoduleState.MaintenanceRequired
This field shall be set according to the Table 301.

Table 301 – SubmoduleState.MaintenanceRequired

Value (hexadecimal)	Meaning	Description
0x00	No MaintenanceRequired available	No Channel of the Submodule requires maintenance
0x01	MaintenanceRequired available	At least one Channel of the Submodule requires maintenance

Bit 5: SubmoduleState.MaintenanceDemanded
This field shall be set according to the Table 302.

Table 302 – SubmoduleState.MaintenanceDemanded

Value (hexadecimal)	Meaning	Description
0x00	No MaintenanceDemanded available	No Channel of the Submodule demandes maintenance
0x01	MaintenanceDemanded available	At least one Channel of the Submodule demandes maintenance

Bit 6: SubmoduleState.DiagInfo
This field shall be set according to the Table 303.

Table 303 – SubmoduleState.DiagInfo

Value (hexadecimal)	Meaning	Description
0x00	No DiagnosisData available	There is no DiagnosisData available/stored for this submodule
0x01	DiagnosisData available	There is DiagnosisData available for this submodule: It can be read with the corresponding records

Bit 7 – 10: SubmoduleState.ARInfo
This field shall be set according to the Table 304.

Table 304 – SubmoduleState.ARInfo

Value (hexadecimal)	Meaning	Description
0x00	Own	This AR is owner of the submodule
0x01	ApplicationReadyPending (ARP)	This AR is owner of the submodule but it is blocked. e.g. parameter checking pending
0x02	Superordinated Locked (SO)	This AR is not owner of the submodule. It is blocked by superordinated means
0x03	Locked By IO Controller (IOC)	This AR is not owner of the submodule. It is owned by an other IOAR
0x04	Locked By IO Supervisor (IOS)	This AR is not owner of the submodule. It is owned by an other IOSAR
0x05 – 0x0F	Reserved	Reserved

Bit 11 – 14: SubmoduleState.IdentInfo
This field shall be set according to the Table 305.

Table 305 – SubmoduleState.IdentInfo

Value (hexadecimal)	Meaning
0x00	OK
0x01	Substitute (SU)
0x02	Wrong (WR)
0x03	NoSubmodule (NO)
0x04 – 0x0F	Reserved

Bit 15: SubmoduleState.FormatIndicator

This field shall be set according to the Table 306.

Table 306 – SubmoduleState.FormatIndicator

Value (hexadecimal)	Meaning	Use
0x00	Coding uses SubmoduleState.Detail	Shall be supported by an IO controller and IO supervisor
0x01	Coding uses SubmoduleState.IdentInfo, .ARInfo and .AddInfo	Shall be used by an IO device, IO controller and IO supervisor

6.2.4.70.2 Coding if SubmoduleState.FormatIndicator == 0

Bit 0 – 14: SubmoduleState.Detail
This field shall be set according to the Table 307.

Table 307 – SubmoduleState.Detail

Value (hexadecimal)	Meaning	Use
0x0000	No submodule	
0x0001	Wrong submodule	
0x0002	Locked by IO controller	
0x0003	Reserved	
0x0004	Application ready pending	Shall only be used in conjunction with IOXControl request (application ready)
0x0005	Reserved	
0x0006	Reserved	
0x0007	Substitute	Submodule is not the same as requested – but compatible and the IO device was able to adapt In this case the IO device has to adapt (e.g. to new input or output length)
0x0008 – 0x7FFF	Reserved	

NOTE The field SubmoduleState is only responded if there is a difference between expected and real configuration data. SubmoduleState "GOOD" is not defined because for such submodules no status is reported within the ExpectedSubmoduleRes.

Bit 15: SubmoduleState.FormatIndicator
This field shall be set according to the Table 306.

6.2.4.71 Coding of the field SubmoduleProperties

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: SubmoduleProperties.Type
This field shall be set according to the Table 308.

Table 308 – SubmoduleProperties.Type

Value (hexadecimal)	Meaning	Use
0x00	No input and no output data	Submodule without IO data, treated as input without data (for IOCS/IOPS), one Input DataDescription Block follows
0x01	Input data	Submodule with input data, one Input DataDescription Block follows
0x02	Output data	Submodule with output data, one Output DataDescription Block follows
0x03	Input and output data	Submodule with input and output data, one Input and one Output DataDescription Block follows

Bit 2: SubmoduleProperties.SharedInput
This field shall be set according to Table 309.

Table 309 – SubmoduleProperties.SharedInput

Value (hexadecimal)	Meaning	Use
0x00	IO controller	IO controller Can be used together with all possible values of SubmoduleProperties.Type.
0x01	IO controller shared	Only the shared IO controller shall use this value. The shared IO controller does not receive alarms and is restricted to read access. The value shall not be used together with SubmoduleProperties.Type==0x02

Bit 3: SubmoduleProperties.ReduceInputSubmoduleDataLength
This field shall be set according to Table 310.

Table 310 – SubmoduleProperties.ReduceInputSubmoduleDataLength

Value (hexadecimal)	Meaning	Use
0x00	Expected	Use expected input SubmoduleDataLength for I-CR
0x01	Zero	Reduce input SubmoduleDataLength to zero for I-CR

Bit 4: SubmoduleProperties.ReduceOutputSubmoduleDataLength
This field shall be set according to Table 311.

Table 311 – SubmoduleProperties.ReduceOutputSubmoduleDataLength

Value (hexadecimal)	Meaning	Use
0x00	Expected	Use expected output SubmoduleDatalength for O-CR
0x01	Zero	Reduce output SubmoduleDatalength to zero for O-CR

Bit 5: SubmoduleProperties.DiscardIOXS
This field shall be set according to Table 312.

Table 312 – SubmoduleProperties.DiscardIOXS

Value (hexadecimal)	Meaning	Use
0x00	Expected	Merge all expected IOXS for this submodule in the frames of the corresponding CRs
0x01	Zero	Discard all expected IOXS for this submodule in the frames of the corresponding CRs

Bit 6 – 7: SubmoduleProperties.reserved_1
This field shall be set according to 3.7.3.2.

Bit 8 – 15: SubmoduleProperties.reserved_2
This field shall be set to zero.

6.2.4.72 Coding of the field ModuleProperties

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: ModuleProperties.reserved_1
This field shall be set according to 3.7.3.2.

Bit 8 – 15: ModuleProperties.reserved_2
This field shall be set to zero.

6.2.4.73 Coding of the field SubstitutionMode

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 313.

Table 313 – SubstitutionMode

Value (hexadecimal)	Meaning	Use
0x0000	ZERO	The outputs are set to zero or inactive
0x0001	Last value	Hold the last valid application value
0x0002	Replacement value	Set the output to the configured replacement value
0x0003 – 0x00FF	Reserved	
0x0100 – 0x01FF	Reserved for profiles	May be used in profile specification
0x0200 – 0xFFFF	Reserved	

6.2.4.74 Coding of the field SubstituteActiveFlag

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 314.

Table 314 – SubstituteActiveFlag

Value (hexadecimal)	Meaning	Use
0x0000	Operation	The outputs are set according to normal control process (substitute inactive)
0x0001	Substitute	The outputs are set according to substitute function (substitute active)
0x0002 – 0xFFFF	Reserved	

6.2.4.75 Coding of the field InitiatorUDPRTPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 315.

Table 315 – InitiatorUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	Reserved	
0x0400 – 0xBFFF	Usable	
0x8892	Default (well known)	Recommended
0xC000 – 0xFFFF	Usable	Recommended

6.2.4.76 Coding of the field ResponderUDPRTPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 316.

Table 316 – ResponderUDPRTPort

Value (hexadecimal)	Meaning	Use
0x0000 – 0x03FF	Reserved	
0x0400 – 0xBFFF	Usable	
0x8892	Default (well known)	Recommended
0xC000 – 0xFFFF	Usable	Recommended

6.2.4.77 Coding of the field InitiatorRPCServerPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 317.

Table 317 – InitiatorRPCServerPort

Value (hexadecimal)	Meaning
0x0000 – 0x03FF	Reserved
0x0400 – 0xFFFF	Usable
0xC000 – 0xFFFF	Recommended according to IANA

6.2.4.78 Coding of the field ResponderRPCServerPort

This field shall be coded as data type Unsigned16. This field shall be coded with the values according to Table 318.

Table 318 – ResponderRPCServerPort

Value (hexadecimal)	Meaning
0x0000 – 0x03FF	Reserved
0x0400 – 0xFFFF	Usable
0xC000 – 0xFFFF	Recommended according to IANA

6.2.4.79 Coding of the field LocalAlarmReference

This field shall be coded as data type Unsigned16. This field shall contain the value of the reference of the IO controller within AlarmCRBlockReq. This field shall contain the value of the reference of the IO device within AlarmCRBlockRes and ARData.

6.2.4.80 Coding of the field RemoteAlarmReference

This field shall be coded as data type Unsigned16. This field shall contain the value of the reference of the IO device within ARData.

6.2.4.81 Coding of the field MaxAlarmDataLength

This field shall be coded as data type Unsigned16. This field shall contain the maximal value of the AlarmNotification-PDU. The allowed values shall be in the range from 200 to 1 432.

6.2.4.82 Coding of the field ParameterServerProperties

This field shall be coded as data type Unsigned32. This field is reserved for future use.

6.2.4.83 Coding of the I&M Records

6.2.4.83.1 General

All data with type VisibleString shall be left justified. If the text is shorter than the defined string length, the gap shall be filled with blanks.

6.2.4.83.2 Coding of the field IM_Serial_Number

This field shall be coded as data type VisibleString[16]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 16 octets.

6.2.4.83.3 Coding of the field IM_Hardware_Revision

This field shall be coded as data type Unsigned16 with the values according to Table 319.

Table 319 – IM_Hardware_Revision

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Hardware revision

6.2.4.83.4 Coding of the field IM_SWRevision_Functional_Enhancement

This field shall be coded as data type Unsigned8 with the values according to Table 320.

Table 320 – IM_SWRevision_Functional_Enhancement

Value (hexadecimal)	Meaning
0x00 – 0xFF	Functional Enhancement

6.2.4.83.5 Coding of the field IM_SWRevision_Bug_Fix

This field shall be coded as data type Unsigned8 with the values according to Table 321.

Table 321 – IM_SWRevision_Bug_Fix

Value (hexadecimal)	Meaning
0x00 – 0xFF	Bug Fix

6.2.4.83.6 Coding of the field IM_SWRevision_Internal_Change

This field shall be coded as data type Unsigned8 with the values according to Table 322.

Table 322 – IM_SWRevision_Internal_Change

Value (hexadecimal)	Meaning
0x00 – 0xFF	Internal Change

6.2.4.83.7 Coding of the field IM_Revision_Counter

This field shall be coded as data type Unsigned16 with the values according to Table 323.

Table 323 – IM_Revision_Counter

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Revision Counter

6.2.4.83.8 Coding of the field IM_Profile_ID

This field shall be coded as data type Unsigned16 with the values according to Table 324.

Table 324 – IM_Profile_ID

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Using shall be defined by profiles, number shall be uniquely administered.

NOTE 1 The IM Profile ID is assigned by PROFIBUS International (PI).

NOTE 2 Depends on 6.2.3.1

6.2.4.83.9 Coding of the field IM_Profile_Specific_Type

This field shall be coded as data type Unsigned16 with the values according to Table 325.

Table 325 – IM_Profile_Specific_Type

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Using shall be defined by profiles and will be treated as a subversion of the profile

6.2.4.83.10 Coding of the field IM_Version_Major

This field shall be coded as data type Unsigned8 with the values according to Table 326.

Table 326 – IM_Version_Major

Value (hexadecimal)	Meaning
0x00	Reserved
0x01	Shall set in this version
0x02 – 0xFF	Reserved

6.2.4.83.11 Coding of the field IM_Version_Minor

This field shall be coded as data type Unsigned8 with the values according to Table 327.

Table 327 – IM_Version_Minor

Value (hexadecimal)	Meaning
0x01	Shall set in this version
0x00, 0x02 – 0xFF	Reserved

6.2.4.83.12 Coding of the field IM_Supported

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: IM_Supported.Profil_specific
This field shall be set related to the profiles.

Bit 1: IM_Supported.I&M1
This field shall be set if the I&M1 record contains data.

Bit 2: IM_Supported.I&M2
This field shall be set if the I&M2 record contains data.

Bit 3: IM_Supported.I&M3
This field shall be set if the I&M3 record contains data.

Bit 4: IM_Supported.I&M4
This field shall be set if the I&M4 record contains data.

Bit 5: IM_Supported.I&M5
This field shall be set if the I&M5 record contains data.

Bit 6: IM_Supported.I&M6
This field shall be set if the I&M6 record contains data.

Bit 7: IM_Supported.I&M7
This field shall be set if the I&M7 record contains data.

Bit 8: IM_Supported.I&M8
This field shall be set if the I&M8 record contains data.

Bit 9: IM_Supported.I&M9
This field shall be set if the I&M9 record contains data.

Bit 10: IM_Supported.I&M10
This field shall be set if the I&M10 record contains data.

Bit 11: IM_Supported.I&M11
This field shall be set if the I&M11 record contains data.

Bit 12: IM_Supported.I&M12
This field shall be set if the I&M12 record contains data.

Bit 13: IM_Supported.I&M13
This field shall be set if the I&M13 record contains data.

Bit 14: IM_Supported.I&M14
This field shall be set if the I&M14 record contains data.

Bit 15: IM_Supported.I&M15
This field shall be set if the I&M15 record contains data.

6.2.4.83.13 Coding of the field IM_Tag_Function

This field shall be coded as data type VisibleString[32]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 32 octets.

6.2.4.83.14 Coding of the field IM_Tag_Location

This field shall be coded as data type VisibleString[22]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 22 octets.

6.2.4.83.15 Coding of the field IM_Date

This field shall be coded as data type VisibleString[16] with the values according to Table 328.

NOTE String format "YYYY-MM-DD HH:MM" is in accordance with the ISO 8601.

Table 328 – IM_Date

Octet	Meaning	Usage
0 – 3	YYYY	Year with four digits
4	"–"	Character dash '–' as separator
5 – 6	MM	Month with two digits
7	"–"	Character dash '–' as separator
8 – 9	DD	Day with two digits
10	" "	Character blank ' ' as separator
11 – 12	HH	Hour with two digits
13	":"	Character colon ':' as separator
14 – 15	MM	Minute with two digits

6.2.4.83.16 Coding of the field IM_Descriptor

This field shall be coded as data type VisibleString[54]. The value shall be set manufacturer specific and shall be filled with blanks if it is shorter than 54 octets.

6.2.4.83.17 Coding of the field IM_Signature

This field shall be coded as data type OctetString[54]. The value shall be set manufacturer specific and shall be filled with zero if it is shorter than 54 octets.

6.2.5 Coding section related to Alarm and Diagnosis PDUs

6.2.5.1 Coding of the field UserStructureIdentifier

This field shall be coded as data type Unsigned16 with the values according to Table 329. This field identifies the structure of the field Data of the AlarmNotification and structure of the field Data of the alarm data.

Table 329 – UserStructureIdentifier

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x7FFF	ManufacturerSpecific	ManufacturerSpecific Diagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification ManufacturerSpecific Diagnosis shall be used in conjunction with following records: All records containing Diagnosis Data In conjunction with other alarm types the usage is manufacturer specific.
0x8000	ChannelDiagnosis	ChannelDiagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification ChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data

Value (hexadecimal)	Meaning	Usage
0x8001	Multiple	Shall only be used in conjunction with Diagnosis, Redundancy, DiagnosisDisappears, MulticastCommunicationMismatch, PortDataChangedNotification, SyncDataChangedNotification and IsochronousModeProblemNotification with data, which comply to the "(BlockHeader, Data*)*" structure. Furthermore, the BlockType shall always correspond to the used AlarmType.
0x8002	ExtChannelDiagnosis	ExtChannelDiagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification ExtChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data.
0x8003	QualifiedChannel-Diagnosis	QualifiedChannelDiagnosis shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification. QualifiedChannelDiagnosis shall be used in conjunction with following Records: All records containing Diagnosis Data.
0x8004 – 0x80FF	Reserved	
0x8100	Maintenance	Maintenance shall be used in conjunction with following AlarmTypes: Diagnosis, Redundancy, DiagnosisDisappears, multicast communication mismatch, Port data change notification, Sync data change notification, isochronous mode problem notification, Network component problem notification, Time data changed notification Furthermore, the AlarmNotification shall only convey this block if the alarm source contains at least one Maintenance Required entry, or one Maintenance Demanded entry, or one Qualifier_x entry. Otherwise the block shall be omitted.
0x8101 – 0x81FF	Reserved	
0x8200	Upload&Retrieval	Upload&Retrieval shall be used in conjunction with upload and retrieval notification.
0x8201	iParameter	iParameter shall be used in conjunction with upload and retrieval notification.
0x8202 – 0x82FF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	
0xA000 – 0xFFFF	Reserved	

6.2.5.2 Coding of the field ChannelErrorType

This field shall be coded as data type Unsigned16 with the values according to Table 330.

Table 330 – ChannelErrorType

Value (hexadecimal)	Meaning	Assigned text
0x0000	Reserved	Unknown error
0x0001	Short circuit	Short circuit
0x0002	Undervoltage	Undervoltage
0x0003	Overvoltage	Overvoltage
0x0004	Overload	Overload
0x0005	Overtemperature	Overtemperature
0x0006	Line break	Line break
0x0007	Upper limit value exceeded	Upper limit value exceeded
0x0008	Lower limit value exceeded	Lower limit value exceeded
0x0009	Error	Error
0x000A	Simulation active	Simulation active
0x000B	Unknown error	Unknown error
0x000C	Unknown error	Unknown error
0x000D	Unknown error	Unknown error
0x000E	Unknown error	Unknown error
0x000F	Manufacturer specific Recommended for “parameter missing”	The channel needs (additional) parameters. No or to less parameters are written
0x0010	Manufacturer specific Recommended for “parametrization fault”	Parametrization fault. Wrong or to many parameters are written
0x0011	Manufacturer specific Recommended for “power supply fault”	Power supply fault
0x0012	Manufacturer specific Recommended for “fuse blown / open”	Fuse blown / open
0x0013	Manufacturer specific Recommended for “communication fault”	Communication fault. Sequence number wrong / sequence wrong
0x0014	Manufacturer specific Recommended for “ground fault”	Ground fault
0x0015	Manufacturer specific Recommended for “reference point lost”	Reference point lost
0x0016	Manufacturer specific Recommended for “process event lost / sampling error”	Process event lost / sampling error
0x0017	Manufacturer specific Recommended for “threshold warning”	Threshold warning
0x0018	Manufacturer specific Recommended for “output disabled”	Output disabled
0x0019	Manufacturer specific Recommended for “safety event”	Safety event
0x001A	Manufacturer specific Recommended for “external fault”	External fault
0x001B	Manufacturer specific	Manufacturer specific
0x001C	Manufacturer specific	Manufacturer specific
0x001D	Manufacturer specific	Manufacturer specific
0x001E	Manufacturer specific	Manufacturer specific

Value (hexadecimal)	Meaning	Assigned text
0x001F	Manufacturer specific Recommended for “temporary fault”	Temporary fault
0x0020 – 0x00FF	Reserved for common profiles	Central administrative [a] number to unambiguous distinguish between common profiles
0x0100 – 0x7FFF	Manufacturer specific	Manufacturer specific
0x8000	Data transmission impossible	Data Transmission Impossible
0x8001	Remote mismatch	Remote Mismatch
0x8002	Media redundancy mismatch	Media Redundancy Mismatch
0x8003	Sync mismatch	Sync Mismatch
0x8004	IsochronousMode mismatch	IsochronousMode Mismatch
0x8005	Multicast CR mismatch	Multicast CR Mismatch
0x8006	Reserved	Reserved
0x8007	Fiber optic mismatch	Information for fiber optic link.
0x8008	Network component function mismatch	Network functionality problems occur
0x8009	Time mismatch	Time master not existent or precision problems
0x800A – 0x8FFF	Reserved	Unknown error
0x9000 – 0x9FFF	Reserved for profiles	Profile specific
0xA000 – 0xFFFF	Reserved	Unknown error

[a] The values of this range are assigned by PROFIBUS International (PI), see <http://www.profinet.com>.

6.2.5.3 Coding of the field ChannelNumber

This field shall be coded as data type Unsigned16 with the values according to Table 331.

Table 331 – ChannelNumber

Value (hexadecimal)	Meaning
0x0000 – 0x7FFF	Manufacturer specific
0x8000	Submodule
0x8001 – 0xFFFF	Reserved

NOTE The ChannelNumber 0x8000 references the submodule itself.

In dependence with the field ChannelProperties.Accumulative the following meaning shall be applied:

- ChannelProperties.Accumulative=1 then the field ChannelNumber shall contain the number of the smallest channel of the effected group
- ChannelProperties.Accumulative=0 then the field ChannelNumber shall contain the number of the effected channel

6.2.5.4 Coding of the field ChannelProperties

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 7: ChannelProperties.Type
This field shall be set according to the Table 332.

Table 332 – ChannelProperties.Type

Value (hexadecimal)	Meaning
0x00	Shall be used if the field ChannelNumber contains the value 0x8000 (submodule). Furthermore, it shall be used if none of the below defined types are appropriate.
0x01	1 Bit
0x02	2 Bit
0x03	4 Bit
0x04	8 Bit
0x05	16 Bit
0x06	32 Bit
0x07	64 Bit
0x08 – 0xFF	Reserved

Bit 8: ChannelProperties.Accumulative
This field shall be set to one if it is an accumulative diagnosis from more then one channel. Otherwise it shall be set to zero.

Bit 9 – 10: ChannelProperties.Maintenance
This field shall be set according to Table 333 and Table 334.

The dependencies with elements of the field ChannelProperties are specified in Table 333 and the dependencies with Alarmnotification and RecordDataRead(DiagnosisData) in Table 334.

Table 333 – Valid combinations within ChannelProperties

Maintenance-Required Bit 9:	Maintenance-Demanded Bit 10:	Specifier	Meaning	Valid with
0x00	0x00	0x00	All subsequent [a] Diagnosis, MaintenanceRequired, MaintenanceDemanded, and QualifiedDiagnosis disappear	ChannelDiagnosis
		0x01	Diagnosis appears	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, and QualifiedChannelDiagnosis
		0x02	Diagnosis disappears	
		0x03	Diagnosis disappears but other remain	
0x01	0x00	0x00	Reserved	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, and QualifiedChannelDiagnosis
		0x01	MaintenanceRequired appears	
		0x02	MaintenanceRequired disappears	
		0x03	MaintenanceRequired disappears but other remain	
0x00	0x01	0x00	Reserved	ChannelDiagnosis, ManufacturerSpecificDiagnosis, ExtChannelDiagnosis, and QualifiedChannelDiagnosis
		0x01	MaintenanceDemanded appears	
		0x02	MaintenanceDemanded disappears	
		0x03	MaintenanceDemanded disappears but other remain	
0x01	0x01	0x00	Reserved	QualifiedChannelDiagnosis
		0x01	QualifiedDiagnosis appears	
		0x02	QualifiedDiagnosis disappears	
		0x03	QualifiedDiagnosis disappears but other remain	

[a] Subsequent means, all ExtChannelErrorTypes for the delivered ChannelErrorType and also the ChannelErrorType itself disappears. Shall be used in AlarmNotification only.

Table 334 – Valid combinations for Alarmnotification and Record-DataRead(DiagnosisData)

Maintenance-Required Bit 9:	Maintenance-Demanded Bit 10:	Specifier	Meaning	Valid with
0x00	0x00	0x00	All subsequent [a] Diagnosis, MaintenanceRequired, MaintenanceDemanded, and QualifiedDiagnosis disappear	Alarmnotification
		0x01	Diagnosis appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	Diagnosis disappears	Alarmnotification RecordDataRead(DiagnosisData) shall only be used in conjunction with ManufacturerSpecific-Diagnosis
		0x03	Diagnosis disappears but other remain	Alarmnotification
0x01	0x00	0x00	Reserved	—
		0x01	MaintenanceRequired appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	MaintenanceRequired disappears	Alarmnotification
		0x03	MaintenanceRequired disappears but other remain	
0x00	0x01	0x00	Reserved	—
		0x01	MaintenanceDemanded appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	MaintenanceDemanded disappears	Alarmnotification
		0x03	MaintenanceDemanded disappears but other remain	
0x01	0x01	0x00	Reserved	—
		0x01	QualifiedDiagnosis appears	Alarmnotification and Record-DataRead(DiagnosisData)
		0x02	QualifiedDiagnosis disappears	Alarmnotification
		0x03	QualifiedDiagnosis disappears but other remain	

[a] Subsequent means, all ExtChannelErrorTypes for the delivered ChannelErrorType and also the ChannelErrorType itself disappears.

Bit 11 – 12: ChannelProperties.Specifier

This field shall be coded with the values according to Table 335. The field ChannelProperties.Specifier is related to the addressed channel, ChannelErrorType, and ExtChannelErrorType. The dependencies with elements of the field ChannelProperties are specified in Table 333.

Table 335 – ChannelProperties.Specifier

Value (hexadecimal)	Meaning	Usage
0x00	All subsequent disappears	See Table 333
0x01	Appears	An event appears and/or exists further
0x02	Disappears	An event disappears and/or exists no longer
0x03	Disappears but other remain	An event disappears but there are other

Bit 13 – 15: ChannelProperties.Direction
This field shall be set according to Table 336.

Table 336 – ChannelProperties.Direction

Value (hexadecimal)	Meaning
0x00	Manufacturer specific
0x01	Input
0x02	Output
0x03	Input/Output
0x04 – 0x07	Reserved

6.2.5.5 Coding of the field ExtChannelErrorType

This field shall be coded as data type Unsigned16. The value of this field depends on the field ChannelErrorType. The values shall be set according to Table 337.

Table 337 – ExtChannelErrorType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Definition is depending on the ChannelErrorType (see tables below).

In conjunction with defined ChannelErrorTypes the ExtChannelErrorType shall be set according to Table 338, Table 339, Table 340, Table 341, Table 342, Table 343, Table 344, Table 345, and Table 346.

Table 338 – ExtChannelErrorType for ChannelErrorType 0 – 0x7FFF

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Accumulative Info	Alarm/diagnosis
0x8001 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 339 – ExtChannelErrorType for ChannelErrorType "Data transmission impossible"

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Port State Mismatch – Link down	Alarm/diagnosis
0x8001	MAU Type Mismatch	Alarm/diagnosis
0x8002	Line Delay mismatch	Alarm/diagnosis
0x8003 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 340 – ExtChannelErrorType for ChannelErrorType "Remote mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Peer Chassis ID mismatch	Alarm/diagnosis
0x8001	Peer Port ID mismatch	Alarm/diagnosis
0x8002	Peer RT_CLASS_3 mismatch	Alarm/diagnosis
0x8003	Peer MAUType mismatch	Alarm/diagnosis
0x8004	Peer MRP domain mismatch	Alarm/diagnosis
0x8005	No peer detected	Alarm/diagnosis
0x8006	Peer MRRT mismatch	Alarm/diagnosis
0x8007	Peer CableDelay mismatch	Alarm/diagnosis
0x8008	Peer PTCP mismatch	Alarm/diagnosis
0x8009 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 341 – ExtChannelErrorType for ChannelErrorType "Media redundancy mismatch"

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Manager role fail	Alarm/diagnosis
0x8001	MRP ring open	Alarm/diagnosis
0x8002	MRRT ring open	Alarm/diagnosis
0x8003	Multiple mananger	Alarm/diagnosis
0x8004 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 342 – ExtChannelErrorType for ChannelErrorType “Sync mismatch” and for ChannelErrorType “Time mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	No Sync Message Received	Alarm/diagnosis
0x8001	Wrong PTCPSubDomainID	Alarm/diagnosis (PTCPoverRTC)
0x8002	Wrong IRDataUUID	Alarm/diagnosis (PTCPoverRTC)
0x8003	Jitter out of Boundary	Alarm/diagnosis
0x8004 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 343 – ExtChannelErrorType for ChannelErrorType “Isochronous mode mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Output Time Failure – Output update missing or out of order	Alarm/diagnosis
0x8001	Input Time Failure	Alarm/diagnosis
0x8002	Master Life Sign Failure – Error in MLS update detected	Alarm/diagnosis
0x8003 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 344 – ExtChannelErrorType for ChannelErrorType “Multicast CR mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Multicast Consumer CR timed out	Alarm/diagnosis
0x8001	Address Resolution Failed	Alarm/diagnosis
0x8002 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 345 – ExtChannelErrorType for ChannelErrorType “Fiber optic mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Power Budget	Alarm/diagnosis
0x8001 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

Table 346 – ExtChannelErrorType for ChannelErrorType “Network component function mismatch”

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001 – 0x7FFF	Manufacturer Specific	Alarm/diagnosis
0x8000	Frame dropped – no resource	Alarm/diagnosis
0x8001 – 0x8FFF	Reserved	
0x9000 – 0x9FFF	Reserved for profiles	Alarm/diagnosis
0xA000 – 0xFFFF	Reserved	

6.2.5.6 Coding of the field ExtChannelAddValue

This field shall be coded as data type Unsigned32. The value 0 shall be used for no information.

6.2.5.6.1 Coding of the field ExtChannelAddValue for ChannelErrorType = 0 – 0x7FFF

This field shall be coded with the values according to Table 347 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0-0x7FFF.

Table 347 – Values for Accumulative Info

Bitposition	Value (hexadecimal)	Meaning
Bit 0	0x00	ChannelNumber is not effected
	0x01	ChannelNumber is effected
Bit 1	0x00	ChannelNumber + 1 is not effected
	0x01	ChannelNumber + 1 is effected
Bit 2	0x00	ChannelNumber + 2 is not effected
	0x01	ChannelNumber + 2 is effected
…	…	…
Bit30	0x00	ChannelNumber + 30 is not effected
	0x01	ChannelNumber + 30 is effected
Bit31	0x00	ChannelNumber + 31 is not effected
	0x01	ChannelNumber + 31 is effected

6.2.5.6.2 Coding of the field ExtChannelAddValue for ChannelErrorType “Fiber optic mismatch”

This field shall be coded with the values according to Table 348 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8007.

Table 348 – Values for “Fiber optic mismatch” – “Power Budget”

Value (hexadecimal)	Meaning
0x00 – 0x3E7	PowerBudget in 0,1dB steps [0..99,9dB]
0x03E8 – 0xFFFFFFFF	Reserved

6.2.5.6.3 Coding of the field ExtChannelAddValue for ChannelErrorType “Network component function mismatch”

This field shall be coded with the values according to Table 349 if the field ExtChannelErrorType contains the value 0x8000 and the field ChannelErrorType contains the value 0x8008.

Table 349 – Values for “Network component function mismatch” – “Frame dropped”

Value (hexadecimal)	Meaning
0x00 – 0x3E7	Number of dropped frames in case of no resource
0x03E8 – 0xFFFFFFFF	Reserved

6.2.5.6.4 Coding of the field ExtChannelAddValue for ChannelErrorType “Remote mismatch”

This field shall be coded with the values according to Table 350 if the field ExtChannelErrorType contains the value 0x8007 and the field ChannelErrorType contains the value 0x8001.

Table 350 – Values for “Remote mismatch” – ”Peer CableDelay mismatch”

Value (hexadecimal)	Meaning	Usage
0x00 – 0x32	Error of measurement	No signaling
0x33 – 0x3B9ACA00	CableDelay difference between peers	Signal deviation
0x3B9ACA01 – 0xFFFFFFFF	Reserved	Reserved

6.2.5.7 Coding of the field QualifiedChannelQualifier

This field shall be coded as data type Unsigned32. The values shall be set according to Table 351. There shall only one bit be set in one QualifiedChannelDiagnosis.

Table 351 – Values for QualifiedChannelQualifier

Bitposition	Value (hexadecimal)	Meaning	Usage
Bit 0	Reserved	Reserved	Reserved
Bit 1	Reserved	Reserved	Reserved
Bit 2	0x00	Qualifier_2 not set	Profile specific
	0x01	Qualifier_2 is set	
Bit 3	0x00	Qualifier_3 not set	
	0x01	Qualifier_3 is set	
…		…	
Bit 30	0x00	Qualifier_30 not set	
	0x01	Qualifier_30 is set	
Bit 31	0x00	Qualifier_31 not set	
	0x01	Qualifier_31 is set	

6.2.5.8 Coding of the field MaintenanceStatus

This field shall be coded as data type Unsigned32. At least one of the bits below shall be set to one in order to convey this alarm block according to Table 352. Otherwise this block shall be omitted.

Table 352 – Values for MaintenanceStatus

Bitposition	Bit name	Value (hexadecimal)	Meaning
Bit 0	MaintenanceRequired	0x00	No maintenance required information available
		0x01	Maintenance required information available
Bit 1	MaintenanceDemanded	0x00	No maintenance demanded information available
		0x01	Maintenance demanded information available
Bit 2	Qualifier_2	0x00	No information available
		0x01	Information available
Bit 3	Qualifier_3	0x00	No information available
		0x01	Information available
…			…
Bit 30	Qualifier_30	0x00	No information available
		0x01	Information available
Bit 31	Qualifier_31	0x00	No information available
		0x01	Information available

The classification of diagnosis, maintenance and qualified is shown in Figure 48.

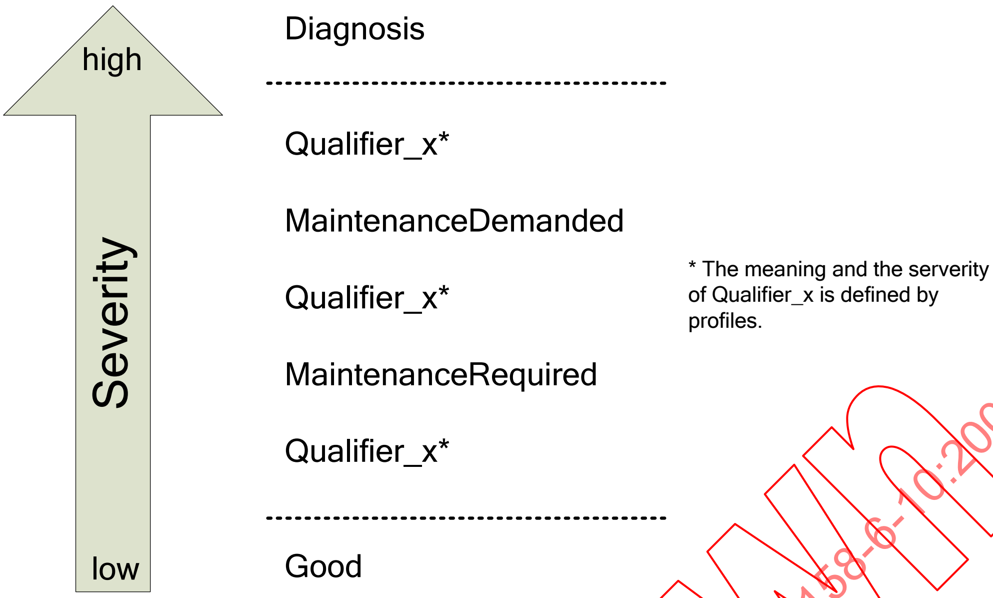

Figure 48 – Classification of diagnosis, maintenance and qualified

6.2.6 Coding section related to upload and retrieval

6.2.6.1 Coding of the field URRecordIndex

This field shall be coded as data type Unsigned32 with the values according to Table 353.

Table 353 – URRecordIndex

Value (hexadecimal)	Meaning
0x00000000 – 0x0000FFFF	Index of the used record
0x00010000 – 0xFFFFFFFF	Reserved

6.2.6.2 Coding of the field URRecordLength

This field shall be coded as data type Unsigned32 with the values according to Table 354.

Table 354 – URRecordLength

Value (hexadecimal)	Meaning
0x00000000 – 0xFFFFFFFF	Length of the used record

6.2.7 Coding section related to iParameter

6.2.7.1 Coding of the field iPar_Req_Header

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.7.2 Coding of the field Max_Segm_Size

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.7.3 Coding of the field Transfer_Index

This field shall be coded as data type Unsigned32 with the values according to IEC 61784-3-3.

6.2.7.4 Coding of the field Total_iPar_Size

This field shall be coded as data type Unsigned32 with the values according to IEC 61784–3–3.

6.2.8 Coding section related to Physical Device Port Data

6.2.8.1 Coding of the field OwnPortID

This field shall be coded as data type OctetString[8] or OctetString[14] according to 4.9.2.3.

6.2.8.2 Coding of the field LengthOwnPortID

This field shall be coded as data type Unsigned8 and the value 8 or 14 according to 4.9.2.3.

6.2.8.3 Coding of the field NumberOfPeers

This field shall be coded as data type Unsigned8.

6.2.8.4 Coding of the field LengthPeerPortID

This field shall be coded as data type Unsigned8.

6.2.8.5 Coding of the field PeerPortID

This field shall be coded as data type OctetString[255].

6.2.8.6 Coding of the field LengthPeerChassisID

This field shall be coded as data type Unsigned8.

6.2.8.7 Coding of the field PeerChassisID

This field shall be coded as data type OctetString[255].

6.2.8.8 Coding of the field LengthOwnChassisID

This field shall be coded as data type Unsigned8.

6.2.8.9 Coding of the field OwnChassisID

This field shall be coded as data type OctetString[255].

6.2.8.10 Coding of the field LineDelay

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: LineDelay.Value
This field shall be set according to the Table 355, Table 356, Figure 18, and Equation (26).

Table 355 – LineDelay.Value with LineDelay.FormatIndicator == 0

Value (hexadecimal)	Meaning
0x00000000	Line delay and cable delay unknown
0x00000001 – 0x7FFFFFFF	Line delay in nanoseconds

Table 356 – LineDelay.Value with LineDelay.FormatIndicator == 1

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x7FFFFFFF	Cable delay in nanoseconds

Bit 31: LineDelay.FormatIndicator
This field shall be set according to the Table 357.

Table 357 – LineDelay.FormatIndicator

Value (hexadecimal)	Meaning	Usage
0x00	LineDelay.Value is coded as line delay	Default, if line delay or cable delay is unknown
0x01	LineDelay.Value is coded as cable delay	Default, if cable delay is known

6.2.8.11 Coding of the field PeerMACAddress

This field shall be coded as data type OctetString[6]. The value of the field PeerMACAddress shall be according to IEEE 802 MAC address.

NOTE Octet 1 contains the Individual/Group Address Bit (lsb).

6.2.9 Coding section related to Physical Device IR Data

6.2.9.1 Coding of the field RxPort

This field shall be coded as data type Unsigned8.This field shall be coded with the values according to Table 358.

Table 358 – RxPort

Value (hexadecimal)	Meaning
0x00	Local interface
0x01	Port 1
0x02	Port 2
…	…
0xFF	Port 255

6.2.9.2 Coding of the field NumberOfTxPortGroups

This field shall be coded as data type Unsigned8 and shall be set according to Table 359. This field shall only count the successing TxPortGroupArray entries.

Table 359 – NumberOfTxPortGroups

Value (hexadecimal)	Meaning
0x01, 0x03, 0x05, 0x07, 0x09, 0x0A, 0x0C, 0x0F	Allowed values
0x11, 0x13, 0x15, 0x17, 0x19, 0x1A, 0x1C, 0x1F	Allowed values
0x21	Allowed values
other	Reserved

6.2.9.3 Coding of the field TxPortGroupArray

The field TxPortGroupArray is an array of octets that shall contain at least one and at most 33 octets referred to as TxPortGroup octet. A TxPortGroup octet shall consist of at least one and at most 8 TxPort entries referred to as TxPortGroup entry 0 to TxPortGroup entry 7. Therefore, the number of TxPortGroup octet corresponds to the number of ports within a device and shall be calculated as follows

$$N = M_{highest} \text{ DIV } 8 + 1 \quad (54)$$

where

N is the number of TxPortGroup octets or the number of array elements,

$M_{highest}$ is the highest number of TxPorts within a device (maximum 255).

The last TxPortGroup octet (octet N) may not contain all 8 TxPort entries. If $M_{highest}$ MOD 8 ≠ 7 the octet N is not fully filled and the remaining bits shall be set to zero referred to as Padding Bits.

NOTE The term DIV stands for division without rest. The term MOD stands for the rest of the division.

The TxPortGroup of the devices TxPorts shall be structured in ascending order without gaps. The coding of this field shall be according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0: TxPortEntry_0
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 0 is present. A padding bit shall be used if no TxPort is present.

Table 360 – TxPortEntry

Value (hexadecimal)	Meaning
0x00	Transmission off
0x01	Transmission on

The TxPort of local injection is always placed in TxPortEntry_1 of the TxPortGroup octet number one.

Bit 1: TxPortEntry_1
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 1 is present. A padding bit shall be used if no TxPort is present.

Bit 2: TxPortEntry_2
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 2 is present. A padding bit shall be used if no TxPort is present.

Bit 3: TxPortEntry_3
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 3 is present. A padding bit shall be used if no TxPort is present.

Bit 4: TxPortEntry_4
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 4 is present. A padding bit shall be used if no TxPort is present.

Bit 5: TxPortEntry_5
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 5 is present. A padding bit shall be used if no TxPort is present.

Bit 6: TxPortEntry_6
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 6 is present. A padding bit shall be used if no TxPort is present.

Bit 7: TxPortEntry_7
This bit shall be set with the values according to Table 360 if the TxPort with the number that meets m MOD 8 = 7 is present. A padding bit shall be used if no TxPort is present.

where m is the number of the current TxPort with 1 ≤ m ≤ N.

6.2.9.4 Coding of the field FrameDetails

This field shall be coded according to 3.7.3.3 and the individual bits shall have the following meaning:

Bit 0 – 1: FrameDetails.SyncFrame
This field shall be coded with the values according to Table 361.

Table 361 – FrameDetails.SyncFrame

Value (hexadecimal)	Meaning
0x00	No sync frame
0x01	Primary sync frame
0x02	Secondary sync frame
0x03	Reserved

Bit 2 – 3: FrameDetails.MeaningFrameSendOffset
This field shall be coded with the values according to Table 362.

Table 362 – FrameDetails.MeaningFrameSendOffset

Value (hexadecimal)	Meaning
0x00	Field FrameSendOffset specifies the point of time for receiving or transmittimg a frame
0x01	Field FrameSendOffset specifies the beginning of the RT_CLASS_3 interval within a phase
0x02	Field FrameSendOffset specifies the ending of the RT_CLASS_3 interval within a phase
0x03	Reserved

Bit 4 – 7: FrameDetails.reserved
This field shall be set to zero.

6.2.9.5 Coding of the field AdjustProperties

This field shall be coded as data type Unsigned16 with the value zero.

6.2.9.6 Coding of the field MAUType

This field shall be coded as data type Unsigned16 with the values according to Table 363 and to Table 364.

Table 363 – MAUType

Value (hexadecimal)	Meaning	Usage
0x0000 – 0x0004	Reserved	
0x0005	10BASET	PDPortDataReal
0x0006-0x0009	Reserved	
0x000A	10BASETXHD	PDPortDataReal
0x000B	10BASETXFD	PDPortDataReal
0x000C	10BASEFLHD	PDPortDataReal
0x000D	10BASEFLFD	PDPortDataReal
0x000F	100BASETXHD	PDPortDataReal
0x0010	100BASETXFD (Default)	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0011	100BASEFXHD	PDPortDataReal
0x0012	100BASEFXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0013 – 0x0014	Reserved	
0x0015	1000BASEXHD	PDPortDataReal
0x0016	1000BASEXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0017	1000BASELXHD	PDPortDataReal
0x0018	1000BASELXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0019	1000BASESXHD	PDPortDataReal
0x001A	1000BASESXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x001B – 0x001C	Reserved	
0x001D	1000BASETHD	PDPortDataReal
0x001E	1000BASETFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x001F	10GigBASEFX	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0020 – 0x002D	Reserved	
0x002E	100BASELX10	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x002F – 0x0035	Reserved	
0x0036	100BASEPXFD	PDPortDataReal, PDPortDataAdjust, PDPortDataCheck
0x0037 – 0xFFFF	Reserved	

Table 364 – Valid combinations between MAUType and PortState

PortState	MAUType	Usage
Up (ready to pass packets)	10BASET	PDPortDataReal
	10BASETXHD	
	10BASETXFD	
	10BASEFLHD	
	10BASEFLFD	
	100BASETXHD	
	100BASETXFD (Default)	
	100BASEFXHD	
	100BASEFXFD	
	1000BASEXHD	
	1000BASEXFD	
	1000BASELXHD	
	1000BASELXFD	
	1000BASESXHD	
	1000BASESXFD	
	1000BASETHD	
	1000BASETFD	
	10GigBASEFX	
	100BASELX10	
	100BASEPXFD	
Down	Reserved	
Testing (in some test mode)		
Unknown (status can not determined)		
Reserved		

6.2.9.7 Coding of the field CheckSyncMode

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: CheckSyncMode.CableDelay
This field shall be set according to the Table 365.

Table 365 – CheckSyncMode.CableDelay

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check cable delay difference between local and remote measured cable delay versus 50 ns.

Bit 1: CheckSyncMode.SyncMaster
This field shall be set according to the Table 366.

Table 366 – CheckSyncMode.SyncMaster

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check PTCP_MasterSourceAddress between local and remote using LLDP_PNIO_PTCPSTATUS.

Bit 2 – 15: CheckSyncMode.reserved
This field shall be set according to 3.7.3.2.

6.2.9.8 Coding of the field MAUTypeMode

The coding of this field shall be according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0: MAUTypeMode.Check
This field shall be set according to the Table 367.

Table 367 – MAUTypeMode.Check

Value (hexadecimal)	Meaning	Usage
0x00	OFF	No check
0x01	ON	Check MAU type difference between local and remote detected value.

Bit 1 – 15: MAUTypeMode.reserved
This field shall be set according to 3.7.3.2.

6.2.9.9 Coding of the field DomainBoundary

The coding of this field shall be according to 3.7.3.5 and the individual bits shall be coded with the values according to Table 368.

Table 368 – DomainBoundary

Bit	Value	Meaning
0	1	Block the multicast MAC address 01-0E-CF-00-04-00
	0	Do not block the multicast MAC address 01-0E-CF-00-04-00
...	1	Block the multicast MAC address 01-0E-CF-00-04-xx
	0	Do not block the multicast MAC address 01-0E-CF-00-04-xx
31	1	Block the multicast MAC address 01-0E-CF-00-04-1F
	0	Do not block the multicast MAC address 01-0E-CF-00-04-1F

6.2.9.10 Coding of the field MulticastBoundary

This field shall be coded as data type Unsigned32. The individual bits shall be coded with the values according to Table 369.

Table 369 – MulticastBoundary

Bit	Value	Meaning
0	1	Block the multicast MAC address 01-0E-CF-00-02-00
	0	Do not block the multicast MAC address 01-0E-CF-00-02-00
…	1	Block the multicast MAC address 01-0E-CF-00-02-xx
	0	Do not block the multicast MAC address 01-0E-CF-00-02-xx
31	1	Block the multicast MAC address 01-0E-CF-00-02-1F
	0	Do not block the multicast MAC address 01-0E-CF-00-02-1F

This shall be applied for the first 32 RT_CLASS_2 multicast addresses from 01-0E-CF-00-02-00 to 01-0E-CF-00-02-1F.

6.2.9.11 Coding of the field PortState

This field shall be coded as data type Unsigned16 with the values according to Table 370.

Table 370 – PortState

Value (hexadecimal)	Meaning	Usage
0x0000	Reserved	
0x0001	Up (ready to pass packets)	PDPortDataReal, CheckPortState
0x0002	Down	PDPortDataReal, AdjustPortState
0x0003	Testing (in some test mode)	PDPortDataReal
0x0004	Unknown (status can not determined)	PDPortDataReal
0x0005 – 0xFFFF	Reserved	

6.2.9.12 Coding of the field MediaType

This field shall be coded as data type Unsigned32 with the values according to Table 371.

Table 371 – MediaType

Value (hexadecimal)	Meaning	Usage
0x00	Unknown	PDPortDataReal
0x01	Copper cable	PDPortDataReal
0x02	Fiber optic cable	PDPortDataReal
0x03	Radio communication	PDPortDataReal
0x04 – 0xFFFFFFFF	Reserved	

6.2.9.13 Coding of the field MaxBridgeDelay

This field shall be coded as data type Unsigned32 according to Table 372. Figure 19 shows the meaning of MaxBridgeDelay.

Table 372 – MaxBridgeDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x3B9AC9FF	From engeneering used bridge delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

6.2.9.14 Coding of the field NumberOfPorts

This field shall be coded as data type Unsigned32. This field shall be coded with the values according to Table 373.

Table 373 – NumberOfPorts

Value (hexadecimal)	Meaning
0x00000000	Reserved
0x00000001 – 0x000000FF	Number of following port entries
0x00000100 – 0xFFFFFFFF	Reserved

6.2.9.15 Coding of the field MaxPortTxDelay

This field shall be coded as data type Unsigned32 according to Table 374. Figure 19 shows the meaning of MaxPortTxDelay.

Table 374 – MaxPortTxDelay

Value (hexadecimal)	Meaning
0x00000000	Unknown
0x00000001 – 0x3B9AC9FF	From engeneering used port transmit delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

6.2.9.16 Coding of the field MaxPortRxDelay

This field shall be coded as data type Unsigned32 according to Table 375. Figure 19 shows the meaning of MaxPortRxDelay.

Table 375 – MaxPortRxDelay

Value (hexadecimal)	Meaning
0x00000000	unknown
0x00000001 – 0x3B9AC9FF	From engeneering used port receive delay for RT_CLASS_3 calculation
0x3B9ACA00 – 0xFFFFFFFF	Reserved

6.2.9.17 Coding of the field Ethertype

This field shall be coded as described in 5.3.2.2.2. In addition to this, the only allowed value is 0x8892.

6.2.10 Coding section related to Physical Sync Data

6.2.10.1 Coding of the field PTCPSubdomainID

This field shall be coded as data type UUID. The value NULL indicates no syncronisation within the Read Real Sync Data service.

6.2.10.2 Coding of the field PTCPLengthSubdomainName

This field shall be coded as data type Unsigned8.

6.2.10.3 Coding of the field PTCPSubdomainName

This field shall be coded as data type OctetString with 1 to 240 octets according to 4.3.1.4.32.1.

NOTE The field PTCPSubdomainName is not terminated by zero.

6.2.10.4 Coding of the field SyncProperties

This field shall be coded according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 1: SyncProperties.Role
This field shall be coded with the values according to Table 376.

Note The information is used for 4.4.1.4.9.

Table 376 – SyncProperties.Role

Value (hexadecimal)	Meaning	Usage
0x00	Reserved	
0x01	External sync	Clock or Time Slave
0x02	Internal sync	Clock or Time Master
0x03	Reserved	

Bit 2 – 7: SyncProperties.reserved
This field shall be set to zero.

Bit 8 – 15: SyncProperties.SyncClass
This field shall be coded with the values according to Table 377 and Table 378.

NOTE The information is used for 4.4.1.4.7.

Table 377 – SyncProperties.SyncClass in conjunction with Clock Master

Value (hexadecimal)	Meaning
0x00	Test Stratum
0x01	Primary (GPS, atom clock)
0x02	Secondary (directly connected to a primary clock)
0x03	Connected to an external time signal (over boundary clocks)
0x04	Not connected to an external time signal
0x05 – 0xFE	Reserved
0xFF	No clock syncronization

Table 378 – SyncProperties.SyncClass in conjunction with Clock Slave

Value (hexadecimal)	Meaning
0x00 – 0xFF	Reserved

6.2.10.5 Coding of the field ReservedIntervalBegin

This field shall be coded as data type Unsigned32. The time base is one ns. This field shall only be valid if the field SyncFrameAddress.MulticastSelection contains the value 0.

6.2.10.6 Coding of the field ReservedIntervalEnd

This field shall be coded as data type Unsigned32. The time base is one ns. This field shall only be valid if the field SyncFrameAddress.MulticastSelection contains the value 0.

6.2.10.7 Coding of the field SyncSendFactor

This field shall be coded as data type Unsigned32. The time base is 31,25 µs. The value range shall be according to Table 379.

Table 379 – SyncSendFactor

Value (hexadecimal)	Meaning	MulticastSelection (Clock)	MulticastSelection (Time)
0x0000	Reserved		
0x0001 – 0x03FF	Optional		
0x0400	Default	Mandatory (32 ms)	Optional (32 ms)
0x0401 – 0x18FFF	Optional		
0x19000	Default		Optional (3,2 s)
0x19001 – 0xF9FFF	Optional		
0xFA000	Default		Mandatory (32 s)
0x000F423F – 0xA4CB7FFF	Optional		
0xA4CB8000	Default		Optional (24 h)
0xA4CB8001 – 0xFFFFFFFF	Reserved		

Each SyncFrameAddress.MulticastSelection shall require its own SyncSendFactor.

6.2.10.8 Coding of the field SyncFrameAddress

This field shall be coded according to 3.7.3.4 and the individual bits shall have the following meaning:

Bit 0 – 4: SyncFrameAddress.MulticastSelection
This field shall be coded with the five least significant bits of the used multicast address of the sync message for a RTASyncPDU according Table 380.

Table 380 – SyncFrameAddress.MulticastSelection for RTASyncPDU

Value (hexadecimal)	Meaning
0x00	01-0E-CF-00-04-00
0x01	01-0E-CF-00-04-01
…	…
0x1F	01-0E-CF-00-04-1F

This field shall be coded for a RTCSyncPDU according table Table 381.

Table 381 – SyncFrameAddress.MulticastSelection for RTCSyncPDU

Value (hexadecimal)	Meaning
0x00	01-0E-CF-00-01-02
0x01-0x1F	Reserved

Bit 5: SyncFrameAddress.PDUType
This field shall be coded with the values according to Table 382.

Table 382 – SyncFrameAddress.PDUType

Value (hexadecimal)	Meaning
0x00	PTCP-RTCSyncPDU
0x01	PTCP-RTASyncPDU

Bit 6 – 15: SyncFrameAddress.reserved
This field shall be set to zero.

6.2.10.9 Coding of the field PTCPTimeoutFactor

This field shall be coded as data type Unsigned16. The time base is the value of the field SyncSendFactor. The value range shall be according to Table 383.

Table 383 – PTCPTimeoutFactor

Value (hexadecimal)	Meaning
0x0000	Disabled
0x0001 – 0x0002	Optional
0x0003	Default, mandatory
0x0004 – 0x000F	Mandatory
0x0010 – 0x01FF	Optional
0x0200 – 0xFFFF	Reserved

Each SyncFrameAddress.MulticastSelection shall require its own PTCPTimeoutFactor. The Timeout shall be calculated according to Equation (55).

$$\text{Timeout} = \text{PTCPTimeoutFactor} \times \text{SyncSendFactor} \times 31{,}25\ \mu\text{s} \qquad (55)$$

6.2.10.10 Coding of the field PLLWindow

This field shall be coded as data type Unsigned32. The time base is one ns. The value range shall be according to Table 384.

Table 384 – PLLWindow

Value (hexadecimal)	Meaning	MulticastSelection (Clock)	MulticastSelection (Time)
0x00	Disabled		
0x0001 – 0x03E7	Optional		
0x03E8	Default	Mandatory (1 µs)	Optional (1 µs)
0x03E9 – 0x270F	Optional		
0x2710	Default	Optional (10 µs)	Optional (10 µs)
0x2710 – 0x000F423F	Optional		
0x000F4240	Default	Optional (1 ms)	Mandatory (1 ms)
0x000F423F – 0x98967F	Optional		
0x989680	Default	Optional (10 ms)	Optional (10 ms)
0x989681 – 0xFFFFFFFF	Reserved		

Each SyncFrameAddress.MulticastSelection shall require its own PLLWindow.

The definition of the PLLWindow is shown in Figure 49.

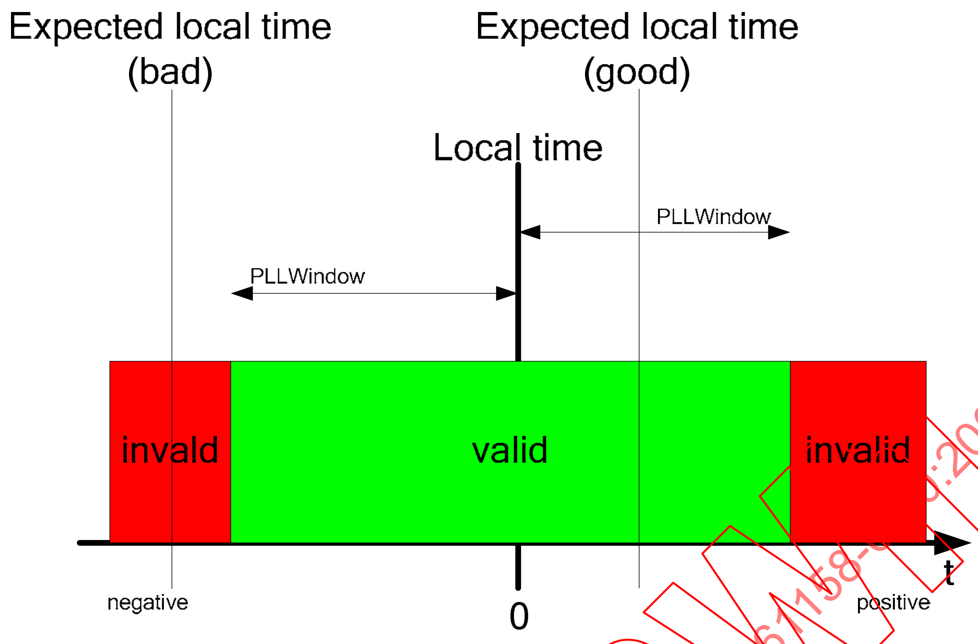

Figure 49 – Definition of PLL window

6.2.11 Coding section related to Isochrone Mode Data

6.2.11.1 Coding of the field TimeDataCycle

This field shall be coded as data type Unsigned16. The time base is 31,25 µs. The value range shall be according to Table 385.

Table 385 – TimeDataCycle

Value (hexadecimal)	Meaning
0x00	Reserved
0x01 – 0x0400	Optional
0x401 – 0xFFFF	Reserved

6.2.11.2 Coding of the field TimeIOInput

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 386.

Table 386 – TimeIOInput

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.3 Coding of the field TimeIOOutput

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 387.

Table 387 – TimeIOOutput

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.4 Coding of the field TimeIOInputValid

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 388.

Table 388 – TimeIOInputValid

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.5 Coding of the field TimeIOOutputValid

This field shall be coded as data type Unsigned32. The time base is ns. The value range shall be according to Table 389.

Table 389 – TimeIOOutputValid

Value (decimal)	Meaning
1 – 32 000 000	Optional
other	Reserved

6.2.11.6 Coding of the field ControllerApplicationCycleFactor

This field shall be coded as data type Unsigned16. The value range shall be according to Table 390.

Table 390 – ControllerApplicationCycleFactor

Value (hexadecimal)	Meaning
0x0000	Reserved
0x0001 – 0x0400	Optional
0x0401 – 0xFFFF	Reserved

6.2.12 Coding section related to Media Redundancy

6.2.12.1 Coding of the field MRP_Role

This field shall be coded as data type Unsigned16 and shall be set according to Table 391.

Table 391 – MRP_Role

Value (hexadecimal)	Meaning
0x0000	Media Redundancy disabled
0x0001	Media Redundancy Client
0x0002	Media Redundancy Manager
0x0003 – 0xFFFF	Reserved

6.2.12.2 Coding of the field MRP_RTMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0: MRP_RTMode.RTClass1_2
This field shall be set according to the Table 392.

Table 392 – MRP_RTMode.RTClass1_2

Value (hexadecimal)	Meaning
0x00	OFF RT_CLASS_1 and RT_CLASS_2 redundancy mode deactivated
0x01	ON RT_CLASS 1 and RT_CLASS 2 redundancy mode activated

Bit 1: MRP_RTMode.RTClass3
This field shall be set according to the Table 393.

Table 393 – MRP_RTMode.RTClass3

Value (hexadecimal)	Meaning
0x00	OFF RT_CLASS_3 redundancy mode deactivated
0x01	ON RT_CLASS_3 redundancy mode activated

Bit 2 – 23: MRP_RTMode.reserved_1
This field shall be set according to 3.7.3.2.

Bit 24 – 31: MRP_RTMode.reserved_2
This field shall be set to zero.

6.2.12.3 Coding of the field MRRT_TSTdefaultT

This field shall be coded as data type Unsigned16, the time base shall be 1 ms, and set according to Table 394.

Table 394 – MRRT_TSTdefaultT

Value (decimal)	Meaning	Usage
0	Reserved	
1 – 99	1 ms – 99 ms	Optional
100	100 ms	Mandatory
101 – 4 000	101 ms – 4 s	Optional
4 001 – 65 535	Reserved	

6.2.12.4 Coding of the field MRP_TOPchgT

This field defines the common point of time that shall be used to invoke the Flush Filtering Data Base service and shall be coded as data type Unsigned16. The value shall be set according to Table 395 with a time base of 10 ms.

Table 395 – MRP_TOPchgT

Value (decimal)	Meaning	Usage
0	0 ms	Clear FDB immediately
1	10 ms	Mandatory
2 – 100	20 ms – 1 s	Optional
101 – 65 535	Reserved	

6.2.12.5 Coding of the field MRP_TOPNRmax

This field shall be coded as data type Unsigned16 and set according to Table 396.

Table 396 – MRP_TOPNRmax

Value (decimal)	Meaning	Usage
0	Reserved	
1	1 iteration	Optional
2	2 iterations	Optional
3	3 iterations	Mandatory (200 ms reconfiguration time)
4	4 iterations	Optional
5	5 iterations	Optional
6 – 65 535	Reserved	

6.2.12.6 Coding of the field MRP_TSTshortT

This field shall be coded as data type Unsigned16 and set according to Table 397.

Table 397 – MRP_TSTshortT

Value (decimal)	Meaning	Usage
0	Reserved	
1 – 9	1 – 9 ms	Optional (short test interval)
10	10 ms	Mandatory (200 ms reconfiguration time)
10 – 500	10 – 500 ms	Optional (short test interval)
501 – 65 535	Reserved	

6.2.12.7 Coding of the field MRP_TSTdefaultT

This field shall be coded as data type Unsigned16 and set according to Table 398.

Table 398 – MRP_TSTdefaultT

Value (decimal)	Meaning	Usage
0	Reserved	
1 – 19	1 – 19 ms	Optional (default test interval)
20	20 ms	Mandatory (200 ms reconfiguration time)
21 – 1 000	21 ms – 1 s	Optional (default test interval)
1 001 – 65 535	Reserved	

6.2.12.8 Coding of the field MRP_TSTNRmax

This field shall be coded as data type Unsigned16 and set according to Table 399.

Table 399 – MRP_TSTNRmax

Value (decimal)	Meaning	Usage
0 – 1	Reserved	
2	2 outstandig test indications cause ring failure	Optional
3	3 outstandig test indications cause ring failure	Mandatory (200 ms reconfiguration time)
4 – 10	4 – 10 outstandig test indications cause ring failure	Optional
11 – 65 535	Reserved	

6.2.12.9 Coding of the field MRRT_TSTNRmax

This field shall be coded as data type Unsigned16 and set according to Table 400.

Table 400 – MRRT_TSTNRmax

Value (decimal)	Meaning	Usage
0 – 1	Reserved	
2	2 outstandig test indications cause ring failure for bumpless redundancy	Optional
3	3 outstandig test indications cause ring failure for bumpless redundancy	Mandatory
4 – 10	4 – 10 outstandig test indications cause ring failure for bumpless redundancy	Optional
11 – 65 535	Reserved	

6.2.12.10 Coding of the field MRP_LNKdownT

This field shall be coded as data type Unsigned16. The coding shall be according to Table 401.

Table 401 – MRP_LNKdownT

Value (decimal)	Meaning	Usage
0	Reserved	
1 – 19	1 – 19 ms Link Down interval	Optional
20	20 ms Link Down interval	Mandatory
21 – 1 000	21 – 1 000 ms Link Down interval	Optional
1 001 – 65 535	Reserved	

6.2.12.11 Coding of the field MRP_LNKupT

This field shall be coded as data type Unsigned16 according to Table 402.

Table 402 – MRP_LNKupT

Value (decimal)	Meaning	Usage
0	Reserved	
1 – 19	1 – 19 ms Link Up interval	Optional
20	20 ms Link Up interval	Mandatory
21 – 1 000	21 – 1 000 ms Link Up interval	Optional
1 001 – 65 535	Reserved	

6.2.12.12 Coding of the field MRP_LNKNRmax

This field shall be coded as data type Unsigned16 according to Table 403.

Table 403 – MRP_LNKNRmax

Value (decimal)	Meaning	Usage
0	Reserved	
1	1 iteration	Optional
2	2 iterations	Optional
3	3 iterations	Optional
4	4 iterations	Mandatory
5	5 iterations	Optional
6 – 65 535	Reserved	

6.2.12.13 Coding of the field MRP_RTState

This field shall be coded as data type Unsigned16 and set according to Table 404.

Table 404 – MRP_RTState

Value (hexadecimal)	Meaning
0x0000	RT media redundancy lost
0x0001	RT media redundancy available
0x0002 – 0xFFFF	Reserved

6.2.12.14 Coding of the field MRP_Check

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0: MRP_Check.MediaRedundancyManager
This field shall be set according to the Table 405.

Table 405 – MRP_Check.MediaRedundancyManager

Value (hexadecimal)	Meaning
0x00	OFF
0x01	ON

Bit 1: MRP_Check.MRP_DomainUUID
This field shall be set according to the Table 406.

Table 406 – MRP_Check.MRP_DomainUUID

Value (hexadecimal)	Meaning
0x00	OFF
0x01	ON Check MRP_DomainUUID vs. LLDP_PNIO_MRPPORTSTATUS

Bit 2 – 23: MRP_Check.reserved_1
This field shall be set according to 3.7.3.2.

Bit 24 – 31: MRP_Check.reserved_2
This field shall be set to zero.

6.2.13 Coding section related to fiber optics

6.2.13.1 Coding of the field VendorBlockType

This field shall be coded as data type Unsigned16 and set according to Table 407.

Table 407 – VendorBlockType

Value (hexadecimal)	Meaning
0x0000 – 0xFFFF	Vendor specific

6.2.13.2 Coding of the field FiberOpticType

This field shall be coded as data type Unsigned32 and set according to Table 408.

Table 408 – FiberOpticType

Value (hexadecimal)	Meaning
0x00000000	No fiber type adjusted
0x00000001	9 µm single mode fiber
0x00000002	50µm multi mode fiber
0x00000003	62,5µm multi mode fiber
0x00000004	SI-POF, NA=0.5
0x00000005	SI-PCF, NA=0.36
0x00000006	LowNA-POF, NA=0.3
0x00000007	GI-POF
0x00000008 – 0x0000007F	Reserved
0x00000080 – 0x000000FF	Vendor specific
0x00000100 – 0xFFFFFFFF	Reserved

6.2.13.3 Coding of the field FiberOpticCableType

This field shall be coded as data type Unsigned32. The coding shall be according to Table 409.

Table 409 – FiberOpticCableType

Value (hexadecimal)	Meaning
0x0000	No cable specified
0x0001	Inside/outside cable, fixed installation
0x0002	Inside/outside cable, flexible installation
0x0003	Outdoor cable, fixed installation
0x0004 – 0xFFFFFFFF	Reserved

6.2.13.4 Coding of the field FiberOpticPowerBudgetType

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: FiberOpticPowerBudgetType.Value
This field shall be set according to the Table 410.

Table 410 – FiberOpticPowerBudgetType.Value

Value (hexadecimal)	Meaning	Usage
0	PowerBudget in 0,1 dB steps	(0 dB): mandatory for maintenance demanded
0x0001 – 0x0013	PowerBudget in 0,1 dB steps	Optional
0x0014	PowerBudget in 0,1 dB steps	(2 dB): mandatory for maintenance required
0x0015 – 0x03E7	PowerBudget in 0,1 dB steps	Optional
0x03E8 – 0x7FFFFFFF	Reserved	Reserved

Bit 31: FiberOpticPowerBudgetType.CheckEnable
This field shall be set according to the Table 410.

Table 411 – FiberOpticPowerBudgetType.CheckEnable

Value (hexadecimal)	Meaning
0x0	OFF
0x1	ON Comparison value is FiberOpticPowerBudgetType.Value

6.2.14 Coding section related network components

6.2.14.1 Coding of the field NCDropBudgetType

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 30: NCDropBudgetType.Value
This field shall be set according to the Table 412.

Table 412 – NCDropBudgetType.Value

Value (hexadecimal)	Meaning	Usage
0	Reserved	Reserved
0x0001 – 0x0002	Number of dropped frames	Optional
0x0003	Number of dropped frames	Mandatory for maintenance required
0x0004 – 0x0009	Number of dropped frames	Optional
0x000A	Number of dropped frames	Mandatory for maintenance demanded
0x000B – 0x03E7	Number of dropped frames	Optional
0x03E8 – 0x7FFFFFFF	Reserved	Reserved

Bit 31: NCDropBudgetType.CheckEnable
This field shall be set according to the Table 413.

Table 413 – NCDropBudgetType.CheckEnable

Value (hexadecimal)	Meaning
0x0	OFF
0x1	ON Comparison value is NCDropBudgetType.Value

The checking of the dropped frames shall be done according to Figure 50 and Figure 51.

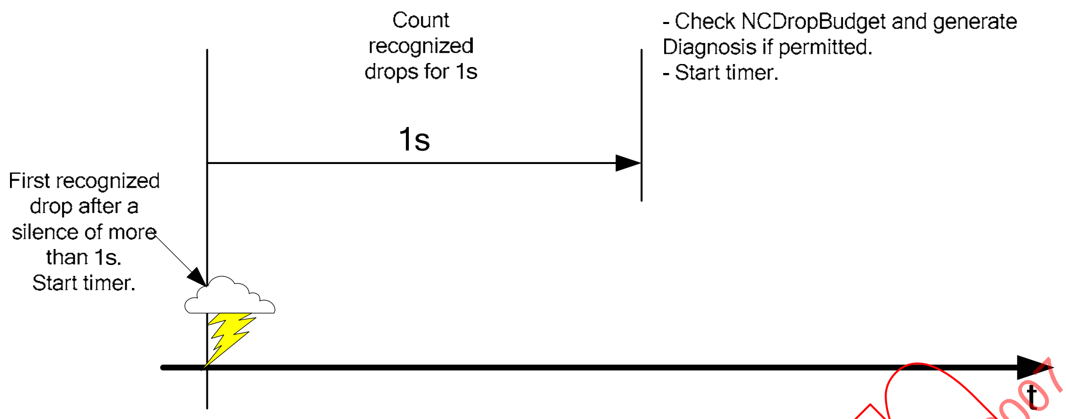

Figure 50 – Detection of dropped frames — appear

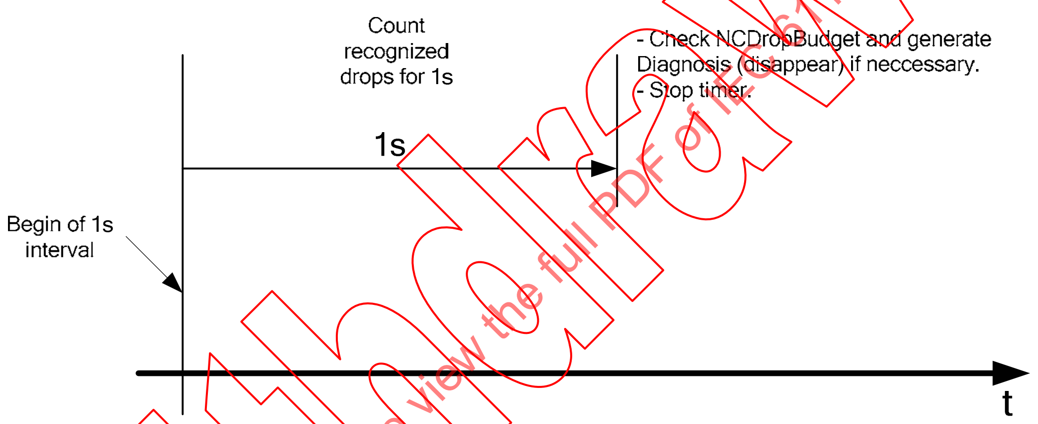

Figure 51 – Detection of dropped frames — disappear

6.2.15 Coding section related to fast start up

6.2.15.1 Coding of the field FSHelloMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 1: FSHelloMode.Mode
This field shall be set according to the Table 414.

Table 414 – FSHelloMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	Send DCP_Hello.req on LinkUp	
0x02	Send DCP_Hello.req on LinkUp after HelloDelay	
0x03	Reserved	Reserved

Bit 2 – 23: FSHelloMode.reserved_1
This field shall be set according to 3.7.3.2.

Bit 24 – 31: FSHelloMode.reserved_2
This field shall be set to zero.

6.2.15.2 Coding of the field FSHelloInterval

This field shall be coded as data type Unsigned32. The coding shall be according to Table 415.

Table 415 – FSHelloInterval

Value (hexadecimal)	Meaning	Usage
0x00000001E	30 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	Default
0x000000032	50 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	
0x000000064	100 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	
0x00000012C	300 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	
0x0000001F4	500 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	
0x0000003E8	1 000 ms Wait this time after the first DCP_Hello.req before conveying a second DCP_Hello.req	
other	Reserved	Reserved

6.2.15.3 Coding of the field FSHelloRetry

This field shall be coded as data type Unsigned32. The coding shall be according to Table 416.

Table 416 – FSHelloRetry

Value (hexadecimal)	Meaning	Usage
0x00000000	Reserved	
0x00000001 – 0x00000002	Number of retransmission of the Hello.req	
0x00000003	Number of retransmission of the Hello.req	Default
0x00000004 – 0x0000000F	Number of retransmission of the Hello.req	
0x00000010 – 0xFFFFFFFF	Reserved	Reserved

6.2.15.4 Coding of the field FSHelloDelay

This field shall be coded as data type Unsigned32. The coding shall be according to Table 417.

Table 417 – FSHelloDelay

Value (hexadecimal)	Meaning	Usage
0x000000000	OFF	Default
0x000000032	50 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	
0x000000064	100 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	
0x0000001F4	500 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	
0x0000003E8	1 000 ms Wait this time after the first LinkUp.ind before conveying a DCP_Hello.req	
other	Reserved	Reserved

6.2.15.5 Coding of the field FSParameterMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 1: FSParameterMode.Mode
This field shall be set according to the Table 418.

Table 418 – FSParameterMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	
0x02	Reserved	Reserved
0x03	Reserved	Reserved

Bit 2 – 23: FSParameterMode.reserved_1
This field shall be set according to 3.7.3.2.

Bit 24 – 31: FSParameterMode.reserved_2
This field shall be set to zero.

6.2.15.6 Coding of the field FSParameterUUID

This field shall be coded as data type UUID. The coding shall be according to Table 419.

Table 419 – FSParameterUUID

Value (UUID)	Meaning
00000000-0000-0000-0000-000000000000	Reserved
00000000-0000-0000-0000-000000000001 – FFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF	Unique UUID for the record data (including physical device data, ...) delivered between IODConnectRes und IODControlReq.

6.2.15.7 Coding of the field FSMode

The coding of this field shall be according to 3.7.3.5 and the individual bits shall have the following meaning:

Bit 0 – 1: FSMode.Mode
This field shall be set according to the Table 420.

Table 420 – FSMode.Mode

Value (hexadecimal)	Meaning	Usage
0x00	OFF	Default
0x01	ON	
0x02	Reserved	Reserved
0x03	Reserved	Reserved

Bit 2 – 23: FSMode.reserved_1
This field shall be set according to 3.7.3.2.

Bit 24 – 31: FSMode.reserved_2
This field shall be set to zero.

6.2.16 PDU checking rules

6.2.16.1 Overview

The following rules shall be applied to check FAL PDUs at the receiver. PDU code checking rules provide redundant information in a condensed way. In case of a contradiction of PDU code checking rules with other parts of the specification, other parts of the specification shall have precedence.

6.2.16.2 IODConnectReq

6.2.16.2.1 ArgsLength

This field shall be checked according to Table 421.

Table 421 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	Okay, check IODConnectReq-PDU
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	RMPM: ArgsLength invalid
IODConnectReq	ARBlockReq	Not first block	RSP-, 0xDB, 0x81, 0x01, 0
IODConnectReq		Case ARProperties.DeviceAccess == 1: other blocks than ARBlockReq	RMPM: Unknown blocks
IODConnectReq	IOCRBlockReq	Case ARProperties.Device-Access == 0: Number of IOCRBlockReq with CRType == "Input CR" == 0 OR Number of IOCRBlockReq with CRType == "Output CR" == 0	RMPM: IOCR missing
IODConnectReq	AlarmCRBlockReq	Case ARProperties.Device-Access == 0: Number of AlarmCRBBlockReq != 1	RMPM: Wrong AlarmCRBlock count

6.2.16.2.2 ARBlockReq

This field shall be checked according to Table 422.

Table 422 – ARBlockReq – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0101	RSP-, 0xDB, 0x81, 0x01, 0
BlockLength	!= 54 + StationNameLength	RSP-, 0xDB, 0x81, 0x01, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x01, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x01, 3
ARType	== reserved OR not equal to one in the GSDML	RSP-, 0xDB, 0x81, 0x01, 4
ARUUID	== NIL	RSP-, 0xDB, 0x81, 0x01, 5
CMInitiatorMacAdd	If (IOCRProperties.RTClass == RT_CLASS_1_UDP) AND (CMInitiatorMacAdd != NULL)	RSP-, 0xDB, 0x81, 0x01, 7
CMInitiatorMacAdd	== multicast address	RSP-, 0xDB, 0x81, 0x01, 7
CMInitiatorObjectUUID	<see next 5 lines>	
.. time low	!= 0xDEA00000	RSP-, 0xDB, 0x81, 0x01, 8
.. time mid	!= 0x6C97	RSP-, 0xDB, 0x81, 0x01, 8
.. time high version	!= 0x11D1	RSP-, 0xDB, 0x81, 0x01, 8
.. Clock[2]	!= 0x82, 0x71	RSP-, 0xDB, 0x81, 0x01, 8
.. Node[6] (contains instance, device, vendor)	Not checked	n. a.
ARProperties.State	== reserved	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.State	If ARType == (IOCARSingle OR IOSAR) AND ARProperties.State != Primary	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.Supervisor-TakeoverAllowed	Don't care	n. a.
ARProperties.Parametrization Server	== External PrmServer	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.DataRate	(==0x02 OR 0x03) AND not suported in GSDML	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.resreved_1	Not checked	n. a.
ARProperties.DeviceAccess	== 1 AND ARType != IOSAR	RSP-, 0xDB, 0x81, 0x01, 9
ARProperties.CompanionAR	== 3	RSP-, 0xDB, 0x81, 0x01, 9
CMInitiatorActivityTimeoutFactor	!= 1 to 1 000 (Base 100 ms)	RSP-, 0xDB, 0x81, 0x01, 10
InitiatorUDPRTPort	== 0 to 0x03FF	RSP-, 0xDB, 0x81, 0x01, 11
StationNameLength	== 0 OR > 240	RSP-, 0xDB, 0x81, 0x01, 12
CMInitiatorStationName	NOT (VisibleString, according to RFC 3490)	RSP-, 0xDB, 0x81, 0x01, 13

6.2.16.2.3 IOCRBlockReq

This field shall be checked according to Table 423.

Table 423 – IOCRBlockReq – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0102	n. a.
BlockLength	!= 42 + NumberOfAPI * (8 + NumberOfIODataObjects * 6 + NumberOfIOCS * 6)	RSP-, 0xDB, 0x81, 0x02, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x02, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x02, 3
IOCRType	!= 0x01 to 0x04	RSP-, 0xDB, 0x81, 0x02, 4
IOCRReference	Is not unique OR (IOCRType == 0x4 AND no correspondig MCRBlock)	RSP-, 0xDB, 0x81, 0x02, 5
LT	Case IOCRProperties.RTClass != 0x04: != 0x8892 Case IOCRProperties.RTClass == 0x04: != 0x0800	RSP-, 0xDB, 0x81, 0x02, 6
IOCRProperties.RTClass	!= 0x01 to 0x04 OR not supported in GSDML	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties.reserved_1	!= 0x0	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties. MediaRedundancy	!= 0x0 OR 0x01 OR not supported in GSDML	RSP-, 0xDB, 0x81, 0x02, 7
IOCRProperties.reserved_2	!= 0x0	RSP-, 0xDB, 0x81, 0x02, 7
DataLength	If (IOCRProperties.RTClass == 0x04 AND DataLength != 12 to 1 440) OR ((IOCRProperties.RTClass == 0x01 OR IOCRProperties.RTClass == 0x02) AND DataLength != 40 to 1 440) OR (IOCRProperties.RTClass == 0x03 AND DataLength != 0 to 1 440)	RSP-, 0xDB, 0x81, 0x02, 8
FrameID	If IOCRType==(MULTICAST_CONSUMER_CR OR MULTICAST_PROVIDER_CR) AND (IOCRProperties.RTClass == 0x01 AND FrameID!=0xF800 to 0xFBFF) OR (IOCRProperties.RTClass == 0x02 AND FrameID!=0xBC00 to 0xBFFF) OR (IOCRProperties.RTClass == 0x03 AND FrameID!=0x0100 to 0x7FFF) OR (IOCRProperties.RTClass == 0x04 AND FrameID!=0xC000 to 0xF7FF)	RSP-, 0xDB, 0x81, 0x02, 9

Parameter	Checking rules	Behaviour on match
FrameID	If IOCRType==INPUT_CR AND (IOCRProperties.RTClass == 0x01 AND FrameID!=0xC000 to 0xF7FF) OR (IOCRProperties.RTClass == 0x02 AND FrameID!=0x8000 to 0xBBFF) OR (IOCRProperties.RTClass == 0x03 AND FrameID!=0x0100 to 0x7FFF) OR (IOCRProperties.RTClass == 0x04 AND FrameID!=0xC000 to 0xF7FF)	RSP-, 0xDB, 0x81, 0x02, 9
SendClockFactor	!=1 to 128 OR not supported in GSDML OR (IOCRProperties.RTClass == 0x3 AND SendClockFactor != LocalClock) OR (IOCRProperties.RTClass == 0x3 AND SendClockFactor * ReducationRatio < LocalClock)	RSP-, 0xDB, 0x81, 0x02, 10
ReductionRatio	If IOCRProperties.RTClass == 0x01 to 0x03 AND (ReductionRatio != 1 to 512 OR not supported in GSDML) OR (ReductionRatio ≥ 256 AND SendClockFactor > 64) OR (ReductionRatio == 512 AND SendClockFactor > 32)	RSP-, 0xDB, 0x81, 0x02, 11
ReductionRatio	If IOCRProperties.RTClass == 0x04 AND (ReductionRatio != 1 to 16 384 OR not supported in GSDML) OR (ReductionRatio ≥ 8 192 AND SendClockFactor > 64) OR (ReductionRatio == 16 384 AND SendClockFactor > 32))	RSP-, 0xDB, 0x81, 0x02, 11
Phase	If Phase == 0x0 OR Phase > ReductionRatio	RSP-, 0xDB, 0x81, 0x02, 12
Sequence	not checked	n. a.
FrameSendOffset	If (FrameSendOffset ≥ SendClockFactor * 31 250 ns) OR ((FrameSendOffset >= 0x003D0900) AND (FrameSendOffset <= 0xFFFFFFFE))	RSP-, 0xDB, 0x81, 0x02, 14
WatchdogFactor	If WatchdogFactor != 0x0001 to 0x1E00 OR (IOCRProperties.RTClass == 0x04 AND WatchdogFactor * ReductionRatio * SendClockFactor * 31,25 > 61 440 000) OR (IOCRProperties.RTClass == 0x01 to 0x03 AND WatchdogFactor * ReductionRatio * SendClockFactor * 31,25 > 1 920 000)	RSP-, 0xDB, 0x81, 0x02, 15

Parameter	Checking rules	Behaviour on match
DataHoldFactor	If DataHoldFactor != 0x0001 to 0x1E00 OR (IOCRProperties.RTClass == 0x04 AND DataHoldFactor * ReductionRatio * SendClockFactor * 31,25 > 61 440 000) OR (IOCRProperties.RTClass == 0x01 to 0x03 AND DataHoldFactor * ReductionRatio * SendClockFactor * 31,25 > 1 920 000)	RSP-, 0xDB, 0x81, 0x02, 16
IOCRTagHeader.IOCRVLANID	Not checked	n. a.
IOCRTagHeader.IOUserPriority	!= IO CR Priority	RSP-, 0xDB, 0x81, 0x02, 17
IOCRMulticastMACAdd	Case IOCRType == 0x03 or 0x04: Is not multicast adress	RSP-, 0xDB, 0x81, 0x02, 18
NumberOfAPI	NumberOfAPI == 0	RSP-, 0xDB, 0x81, 0x02, 19
API	API not in corresponding ExpectedSubmoduleBlockReq	RSP-, 0xDB, 0x81, 0x02, 20
NumberOfIODataObjects	== 0x0 AND NumberOfIOCS = 0x0	RSP-, 0xDB, 0x81, 0x02, 20
SlotNumber	Not in corresponding ExpectedSubmoduleBlockReq	RSP-, 0xDB, 0x81, 0x02, 22
SubslotNumber	Not in corresponding ExpectedSubmoduleBlockReq OR ((corresponding SubmoduleProperties.Type == 0x0 to 0x1) AND (IOCRType == 0x2 OR 0x4)) OR ((corresponding SubmoduleProperties.Type == 0x2) AND (IOCRType == 0x1 OR 0x3))	RSP-, 0xDB, 0x81, 0x02, 23
IODataObjectFrameOffset	>= DataLength OR (IODataObjectFrameOffset + (effective SubmoduleDataLength + effective LengthIOPS of corresponding submodule) >= DataLength) OR (IOCRType != “multicast consumer CR” AND (IODataObjectFrameOffset + (effective SubmoduleDataLength + effective LengthIOPS of corresponding submodule) overlap with other IODataObjects or IOCSes))	RSP-, 0xDB, 0x81, 0x02, 24
NumberOfIOCS	== 0x0 AND NumberOfIODataObjects == 0x0	RSP-, 0xDB, 0x81, 0x02, 25
SlotNumber	Not in corresponding ExpectedSubmoduleBlockReq	RSP-, 0xDB, 0x81, 0x02, 26
SubslotNumber	Not in corresponding ExpectedSubmoduleBlockReq OR ((corresponding SubmoduleProperties.Type == 0x0 to 0x1) AND (IOCRType == 0x1)) OR ((corresponding SubmoduleProperties.Type == 0x2) AND (IOCRType == 0x2))	RSP-, 0xDB, 0x81, 0x02, 27

Parameter	Checking rules	Behaviour on match
IOCSFrameOffset	>= DataLength OR not unique OR (IOCSFrameOffset + (effective LengthIOCS of corresponding submodule) >= DataLength) OR (IOCSFrameOffset + (effective LengthIOCS of corresponding submodule) overlap with other IODataObjects or IOCSes)	RSP-, 0xDB, 0x81, 0x02, 28

6.2.16.2.4 AlarmCRBlockReq

This field shall be checked according to Table 424.

Table 424 – AlarmCRBlockReq – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0103	n. a.
BlockLength	!= 22	RSP-, 0xDB, 0x81, 0x04, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x04, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x04, 3
AlarmCRType	!= 0x0001	RSP-, 0xDB, 0x81, 0x04, 4
LT	Case AlarmCRProperties.Transport == 0x00: != 0x8892 Case AlarmCRProperties.Transport == 0x01: != 0x0800	RSP-, 0xDB, 0x81, 0x04, 5
AlarmCRProperties.Priority	Not checked	n. a.
AlarmCRProperties.Transport	Not checked	n. a.
AlarmCRProperties.reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x04, 6
RTATimoutFactor	!= 0x0001 to 0x0064 optional: AND != 0x0065 to 0xFFFF	RSP-, 0xDB, 0x81, 0x04, 7
RTARetries	!= 3 to 15	RSP-, 0xDB, 0x81, 0x04, 8
LocalAlarmReference	Not checked	n. a.
MaxAlarmDataLength	!= 200 to 1 432	RSP-, 0xDB, 0x81, 0x04, 10
AlarmCRTagHeaderHigh. AlarmCRVLANID	> 0xFFF OR != AlarmCRTagHeaderLow. AlarmCRVLANID	RSP-, 0xDB, 0x81, 0x04, 11
AlarmCRTagHeaderHigh. AlarmUserPriority	!= Alarm CR Priority High	RSP-, 0xDB, 0x81, 0x04, 11
AlarmCRTagHeaderLow. AlarmCRVLANID	> 0xFFF OR != AlarmCRTagHeaderHigh. AlarmCRVLANID	RSP-, 0xDB, 0x81, 0x04, 12
AlarmCRTagHeaderLow. AlarmUserPriority	!= Alarm CR Priority Low	RSP-, 0xDB, 0x81, 0x04, 12

6.2.16.2.5 ExpectedSubmoduleBlockReq

This field shall be checked according to Table 425.

Table 425 – ExpectedSubmoduleBlockReq – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0104	n. a.
BlockLength	!= 4 + NumberOfAPI * (14 + NumberOfSubmodules * (8 + NumberOfSubsequentDataDescriptionBlocks * 6)	RSP-, 0xDB, 0x81, 0x03, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x03, 3
NumberOfAPI	== 0	RSP-, 0xDB, 0x81, 0x03, 4
API	> 0x0 AND not supported in GSDML	RSP-, 0xDB, 0x81, 0x03, 5
SlotNumber	!= 0x0 to 0x7FFF OR not supported in GSDML OR not unique	RSP-, 0xDB, 0x81, 0x03, 6
ModuleIdentNumber	== 0x00000000	RSP-, 0xDB, 0x81, 0x03, 6
ModuleProperties.reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 8
NumberOfSubmodules	== 0	RSP-, 0xDB, 0x81, 0x03, 9
SubslotNumber	Case ARProperties.PullModuleAlarmAllowed (=0): != 0x0001 to 0x7FFF OR not supported in GSDML OR not unique OR not used in at least one IOCR OR == 0x8000 to 0x8FFF AND API != 0 Case ARProperties.PullModuleAlarmAllowed (=1): != 0x0000 to 0x7FFF OR not supported in GSDML OR not unique OR not used in at least one IOCR OR == 0x8000 to 0x8FFF AND API != 0	RSP-, 0xDB, 0x81, 0x03, 10
SubmoduleProperties.Type	== (NO_IO OR INPUT) AND one output description block follows OR == (OUTPUT OR IO) AND one input description block follows	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties. SharedInput	If SubmoduleProperties.Type == (OUTPUT DATA) AND SubmoduleProperties.SharedInput == 0x1	RSP-, 0xDB, 0x81, 0x03, 12
SubmoduleProperties. ReduceInputSubmoduleDataLength	Not checked	n. a.
SubmoduleProperties. ReduceOutputSubmoduleDataLength	Not checked	n. a.
SubmoduleProperties. DiscardIOXS	Not checked	n. a.
SubmoduleProperties. reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 12

Parameter	Checking rules	Behaviour on match
DataDescription.Type	If DataDescription.Type == (0x00 OR 0x03) OR DataDescription.Type == 0x01 AND SubmoduleProperties.Type == 0x02 OR DataDescription.Type == 0x02 AND SubmoduleProperties.Type != (0x02 OR 0x03)	RSP-, 0xDB, 0x81, 0x03, 13
DataDescription.reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x03, 13
SubmoduleDataLength	!= 0 to 1 439	RSP-, 0xDB, 0x81, 0x03, 14
LengthIOPS	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 15
LengthIOCS	!= 0x01	RSP-, 0xDB, 0x81, 0x03, 16

6.2.16.2.6 PrmServerBlock

This field shall be checked according to Table 426.

Table 426 – PrmServerBlock – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0105	n. a.
BlockLength	!= 26 + StationNameLength	RSP-, 0xDB, 0x81, 0x05, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x05, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x05, 3
ParameterServerObjectUUID	Not checked	n. a.
ParameterServerProperties	Not checked	n. a.
CMInitiatorActivityTimeoutFactor	!= 1 to 1 000	RSP-, 0xDB, 0x81, 0x05, 6
StationNameLength	!= 1 to 240	RSP-, 0xDB, 0x81, 0x05, 7
ParameterServerStationName	Not visible string according to RFC 3490	RSP-, 0xDB, 0x81, 0x05, 8

6.2.16.2.7 MCRBlock

This field shall be checked according to Table 427.

Table 427 – MCRBlockReq – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0106	n. a.
BlockLength	!= (12 + StationNameLength + Padding) AND (BlockLength mod 4 == 0)	RSP-, 0xDB, 0x81, 0x06, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x06, 2
BlockVersionLow	!= 0x00	RSP-, 0xDB, 0x81, 0x06, 3
IOCRReference	Is not unique OR Corresponding IOCRType != 0x4	RSP-, 0xDB, 0x81, 0x06, 4
AddressResolutionProperties. Protocol	!= 0x01 OR 0x02	RSP-, 0xDB, 0x81, 0x06, 5
AddressResolutionProperties. Reserved	!= 0x0	RSP-, 0xDB, 0x81, 0x06, 5
AddressResolutionProperties. Factor	!= 0x0001 to 0x0064 optional: AND != 0x0065 to 0xFFFF	RSP-, 0xDB, 0x81, 0x06, 5
MCITimeoutFactor	> 100	RSP-, 0xDB, 0x81, 0x06, 6
StationNameLength	!= 1 to 240	RSP-, 0xDB, 0x81, 0x06, 7
ProviderStationName	Not visible string according to RFC 3490	RSP-, 0xDB, 0x81, 0x06, 8
Padding	Not checked	n. a.

6.2.16.2.8 ARRPCBlock

This field shall be checked according to Table 428.

Table 428 – ARRPCBlockReq – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0107	n. a.
BlockLength	!= 4	RSP-, 0xDB, 0x81, 0x07, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDB, 0x81, 0x07, 2
BlockVersionLow	!= 0x01	RSP-, 0xDB, 0x81, 0x07, 3
InitiatorRPCServerPort	!= 0x0400 to 0xFFFF	RSP-, 0xDB, 0x81, 0x07, 4

6.2.16.3 IODConnectRes

6.2.16.3.1 ArgsLength

6.2.16.3.2 ARRPCBlock

This field shall be checked according to Table 429.

Table 429 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	Okay, check IODConnectRes-PDU
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataResponse	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	CMCTL: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the response	CMCTL: Unknown blocks
IODConnectRes			

6.2.16.3.3 ARBlockRes

This field shall be checked according to Table 430.

Table 430 – ARBlockRes – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8101	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 30	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARType	!= ARBlockReq.ARType	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	== ARBlockReq.ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	== ARBlockReq.SessionKey	n. a.
SessionKey	!= ARBlockReq.SessionKey	ignore
CMResponderMacAdd	Is not unicast	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ResponderUDPRTPort	== 0x0001 to 0x03FF	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.3.4 IOCRBlockRes

This field shall be checked according to Table 431.

Table 431 – IOCRBlockRes – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8102	n. a.
BlockLength	!= 8	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IOCRType	!= IOCRBlockReq.IOCRType	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IOCRReference	!= IOCRBlockReq.IOCRReference	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
FrameID	If (IOCRBlockReq.IOCRType == "Input CR" OR IOCRBlockReq.IOCRType == "Multicast Consumer CR") AND IOCRBlockReq.FrameID != FrameID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.3.5 AlarmCRBlockRes

This field shall be checked according to Table 432.

Table 432 – AlarmCRBlockRes – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8103	n. a.
BlockLength	!= 8	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
AlarmCRType	!= 0x0001	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
LocalAlarmReference	Not checked	n. a.
MaxAlarmDataLength	!= 200 to 1 432	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.3.6 ModuleDiffBlock

This field shall be checked according to Table 433.

Table 433 – ModuleDiffBlock – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8104	n. a.
BlockLength	!= 4 + NumberOfAPI * (6 + NumberOfModules * (12 + NumberOfSubmodules * 8))	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
NumberOfAPIs	== 0	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
API	!= ExpectedSubmoduleBlockReq.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
NumberOfModules	== 0	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	!= ExpectedSubmoduleBlockReq.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ModuleState	== 0x0 to 0x0003	n. a.
ModuleState	!= 0x0 to 0x0003	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
NumberOfSubmodules	Not checked	n. a.
SubslotNumber	!= ExpectedSubmoduleBlockReq.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubmoduleIdentNumber	Not checked	n. a.
SubmoduleState.AddInfo	Not checked	n. a.
SubmoduleState.QualifiedInfo	Not checked	n. a.
SubmoduleState.Maintenance-Required	Not checked	n. a.
SubmoduleState.Maintenance-Demanded	Not checked	n. a.
SubmoduleState.DiagInfo	Not checked	n. a.
SubmoduleState.ARInfo	If SubmoduleState.FormatIndicator == 1 != 0x0 to 0x04	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubmoduleState.IdentInfo	If SubmoduleState.FormatIndicator == 1 != 0x0 to 0x03	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubmoduleState. FormatIndicator	Not checked	n. a.
SubmoduleState.Detail	If SubmoduleState.FormatIndicator == 0 != (0x0 to 0x02 OR 0x04 OR 0x07)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.4 IODControlReq

6.2.16.4.1 ArgsLength

This field shall be checked according to Table 434.

Table 434 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	Okay, check IODControlReq-PDU
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the request	RMPM: Unknown blocks
IODControlReq			

6.2.16.4.2 ControlBlockConnect request

This field shall be checked according to Table 435.

Table 435 – ControlBlockConnect – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0110	RSP-, 0xDD, 0x81, 0x14, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x14, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x14, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x14, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x14, 4
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x14, 6
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x14, 7
ControlCommand	!= 0x0001 (PrmEnd)	RSP-, 0xDD, 0x81, 0x14, 8
ControlBlockProperties.reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x14, 9

6.2.16.4.3 ControlBlockPlug

This field shall be checked according to Table 436.

Table 436 – ControlBlockPlug – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0111	RSP-, 0xDD, 0x81, 0x15, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x15, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x15, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x15, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x15, 4
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x15, 6
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	RSP-, 0xDD, 0x81, 0x15, 7
ControlCommand	!= 0x0001 (PrmEnd)	RSP-, 0xDD, 0x81, 0x15, 8
ControlBlockProperties.reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x15, 9

6.2.16.5 IODControlRes

6.2.16.5.1 ArgsLength

This field shall be checked according to Table 437.

Table 437 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	Okay, check IODControlRes-PDU
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataRespons e	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	CMCTL: ArgsLength invalid
	ArgsLength	General: If one OR more unkonwn blocks are contained in the response	CMCTL: Unknown blocks
IODControlRes			

6.2.16.5.2 ControlBlockConnect

This field shall be checked according to Table 438.

Table 438 – ControlBlockConnect – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8110	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= IODControlReq.ControlBlockConnect. ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODControlReq.ControlBlockConnect. SessionKey	ignore
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.5.3 ControlBlockPlug

This field shall be checked according to Table 439.

Table 439 – ControlBlockPlug – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8111	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= IODControlReq.ControlBlockPlug. ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODControlReq.ControlBlockPlug. SessionKey	ignore
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.6 IOXControlReq

6.2.16.6.1 ArgsLength

This field shall be checked according to Table 440.

Table 440 – ArgsLength check

Parameter		Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	Okay, check IOXControlReq-PDU
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataRequest	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	RMPM: ArgsLength invalid
	ArgsLength	General: If one OR more unkonwn blocks are contained in the request	RMPM: Unknown blocks
IOXControlReq			

6.2.16.6.2 ControlBlockConnect

This field shall be checked according to Table 441.

Table 441 – ControlBlockConnect – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0112	RSP-, 0xDD, 0x81, 0x16, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x16, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x16, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x16, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x16, 4
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x16, 6
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x16, 7
ControlCommand	!= 0x0002 (ApplicationReady)	RSP-, 0xDD, 0x81, 0x16, 8
ControlBlockProperties.reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x16, 9

6.2.16.6.3 ControlBlockPlug

This field shall be checked according to Table 442.

Table 442 – ControlBlockPlug – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0113	RSP-, 0xDD, 0x81, 0x17, 0
BlockLength	!= 28	RSP-, 0xDD, 0x81, 0x17, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDD, 0x81, 0x17, 2
BlockVersionLow	!= 0x00	RSP-, 0xDD, 0x81, 0x17, 3
Padding	!= 0x00	RSP-, 0xDD, 0x81, 0x17, 4
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDD, 0x81, 0x17, 6
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	RSP-, 0xDD, 0x81, 0x17, 7
ControlCommand	!= 0x0002 (ApplicationReady)	RSP-, 0xDD, 0x81, 0x17, 8
ControlBlockProperties.reserved	!= 0x0000	RSP-, 0xDD, 0x81, 0x17, 9

6.2.16.7 IOXControlRes

6.2.16.7.1 ArgsLength

This field shall be checked according to Table 443.

Table 443 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	Okay, check IOXControlRes-PDU
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataResponse	ArgsLength	!= Sum of all BlockLength + Number of Blocks * 4	RMPM: ArgsLength invalid
	ArgsLength	General: If one OR more unkonwn blocks are contained in the response	RMPM: Unknown blocks
IOXControlRes			

6.2.16.7.2 ControlBlockConnect

This field shall be checked according to Table 444.

Table 444 – ControlBlockConnect – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8112	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= IOXControlReq.ControlBlockConnect. ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IOXControlReq.ControlBlockConnect. SessionKey	ignore
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.7.3 ControlBlockPlug

This field shall be checked according to Table 445.

Table 445 – ControlBlockPlug – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8113	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= IOXControlReq.ControlBlockPlug. ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IOXControlReq.ControlBlockPlug. SessionKey	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
AlarmSequenceNumber	!= AlarmSpecifier.SequenceNumber of corresponding AlarmNotification-PDU	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.7.4 ModuleDiffBlock

This field shall be checked according to Table 433.

6.2.16.8 IODReleaseReq

6.2.16.8.1 ArgsLength

This field shall be checked according to Table 446.

Table 446 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	Okay, check IODReleaseReq-PDU
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataRequest	ArgsLength	!= ReleaseBlock.BlockLength + 4	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the request	RMPM: Unknown blocks

	Parameter	Checking rules	Behaviour on match
IODReleaseReq			

6.2.16.8.2 ReleaseBlock

This field shall be checked according to Table 447.

Table 447 – ReleaseBlock – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0114	RSP-, 0xDC, 0x81, 0x28, 0
BlockLength	!= 28	RSP-, 0xDC, 0x81, 0x28, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDC, 0x81, 0x28, 2
BlockVersionLow	!= 0x00	RSP-, 0xDC, 0x81, 0x28, 3
Padding	!= 0x00	RSP-, 0xDC, 0x81, 0x28, 4
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID unknown
SessionKey	!= ARBlockReq.SessionKey	RSP-, 0xDC, 0x81, 0x28, 6
Padding	!= 0x00	RSP-, 0xDC, 0x81, 0x28, 7
ControlCommand	!= 0x0004 (Release)	RSP-, 0xDC, 0x81, 0x28, 8
ControlBlockProperties.reserved	!= 0x0000	RSP-, 0xDC, 0x81, 0x28, 9

6.2.16.9 IODReleaseRes

6.2.16.9.1 ArgsLength

This field shall be checked according to Table 448.

Table 448 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	Okay, check IODReleaseRes-PDU
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataResponse	ArgsLength	!= ReleaseBlock.BlockLength + 4	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the response	RMPM: Unknown blocks
IODReleaseRes			

6.2.16.9.2 ReleaseBlock

This field shall be checked according to Table 449.

Table 449 – ReleaseBlock – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8114	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 28	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ARUUID	!= IODReleaseReq.ReleaseBlock.ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SessionKey	!= IODReleaseReq.ReleaseBlock. SessionKey	ignore
Padding	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlCommand	!= 0x0008 (Done)	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
ControlBlockProperties.reserved	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT

6.2.16.10 IODWriteReq

6.2.16.10.1 ArgsLength

This field shall be checked according to Table 450.

Table 450 – ArgsLength check

Parameter		Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	Okay, check IODWriteReq-PDU
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataRequest	ArgsLength	!= (IODWriteReqHeader. BlockLength + 4 + IODWriteReqHeader. RecordDataLength)	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the request	RMPM: Unknown blocks
IODWriteReq			

6.2.16.10.2 IODWriteReqHeader

This field shall be checked according to Table 451.

Table 451 – IODWriteReqHeader – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0008	RSP-, 0xDF, 0x81, 0x08, 0
BlockLength	!= 60	RSP-, 0xDF, 0x81, 0x08, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDF, 0x81, 0x08, 2
BlockVersionLow	!= 0x00	RSP-, 0xDF, 0x81, 0x08, 3
SeqNumber	Not checked	n. a.
ARUUID	!= ARBlockReq.ARUUID	RMPM: AR UUID unknown
API	> 0x0 AND not supported in GSDML	RSP-, 0xDF, 0x80, 0xB4, 6
SlotNumber	!= 0x0 to 0x7FFF OR not supported in GSDML	RSP-, 0xDF, 0x80, 0xB2, 7
SubslotNumber	!= 0x0 to 0x8FFF OR not supported in GSDML	RSP-, 0xDF, 0x80, 0xB2, 8
Padding	!= 0x0000	RSP-, 0xDF, 0x80, 0xB7, 9
Index	Not supported by application	RSP-, 0xDF, 0x80, 0xB0, 10
RecordDataLength	!= consistent with ArgsLength	RSP-, 0xDF, 0x81, 0x08, 11
RWPadding	Not checked	n. a.

6.2.16.10.3 RecordDataWrite

This field shall be not checked.

6.2.16.11 IODWriteRes

6.2.16.11.1 ArgsLength

This field shall be checked according to Table 452.

Table 452 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
NDRDataRespons e	ArgsLength	!= IODWriteResHeader.BlockLe ngth + 4	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
		General: If one OR more unkonwn blocks are contained in the response	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IODReleaseRes			

6.2.16.11.2 IODWriteResHeader

This field shall be checked according to Table 453.

Table 453 – IODWriteResHeader – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8008	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 60	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SeqNumber	Not checked	n. a
ARUUID	!= IODWriteReq.IODWriteReqHeader. ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
API	!= IODWriteReq.IODWriteReqHeader.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	!= IODWriteReq.IODWriteReqHeader.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubslotNumber	!= IODWriteReq.IODWriteReqHeader.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Index	!= IODWriteReq.IODWriteReqHeader.Index	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
RecordDataLength	Not checked	n. a.
AdditionalValue1	Not checked	n. a.
AdditionalValue2	Not checked	n. a.
PNIOStatus	Not checked	n. a.

6.2.16.12 IODWriteMultipleReq

6.2.16.12.1 ArgsLength

This field shall be checked according to Table 454.

Table 454 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	Okay, check IODWriteMultipleReq-PDU
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	

	Parameter	Checking rules	Behaviour on match
NDRDataRequest	ArgsLength	!= IODWriteReqHeader. BlockLength + 4 + Sum of all IODWriteReqHeader.BlockLength + Number of WriteBlocks * 4	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the request	RMPM: Unknown blocks
IODWriteReq			

6.2.16.12.2 IODWriteReqHeader (multiple)

This field shall be checked according to Table 451. The value of the parameter index shall be 0xE040.

6.2.16.12.3 IODWriteReqHeader

This field shall be checked according to Table 451.

6.2.16.12.4 RecordDataWrite

This field shall be not checked.

6.2.16.12.5 Padding

The number of padding octets shall be 0, 1, 2, and 3 to have 32 bit alignment to the next IODWriteReqHeader. The value of the padding octets is not checked.

6.2.16.13 IODWriteMultipleRes

6.2.16.13.1 ArgsLength

This field shall be checked according to Table 455.

Table 455 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	
	RPCOperationNmb	== 3 (Write)	Okay, check IODWriteMultipleRes-PDU
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	
NDRDataResponse	ArgsLength	!= IODWriteResHeader. BlockLength + 4 + Sum of all IODWriteResHeader.BlockLength + Number of IODWriteResHeaderBlocks * 4	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the response	RMPM: Unknown blocks
IODWriteMultipleRes			

6.2.16.13.2 IODWriteResHeader (multiple)

This field shall be checked according to Table 453. The value of the parameter index shall be 0xE040.

6.2.16.13.3 IODWriteResHeader

This field shall be checked according to Table 453.

6.2.16.14 IODReadReq

6.2.16.14.1 ArgsLength

This field shall be checked according to Table 450.

Table 456 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
RPCHeader	RPCOperationNmb	== 0 (Connect)	
	RPCOperationNmb	== 1 (Release)	
	RPCOperationNmb	== 2 (Read)	Okay, check IODReadReq-PDU
	RPCOperationNmb	== 3 (Write)	
	RPCOperationNmb	== 4 (Control)	
	RPCOperationNmb	== 5 (Read Implicit)	Okay, check IODReadReq-PDU
NDRDataRequest	ArgsLength	!= IODReadReqHeader.BlockLength + 4 OR != IODReadReqHeader.BlockLength + 4 + RecordDataReadQuery.BlockLength + 4	RMPM: ArgsLength invalid
		General: If one OR more unkonwn blocks are contained in the request	RMPM: Unknown blocks
IOReadReq			

6.2.16.14.2 IODReadReqHeader

This field shall be checked according to Table 457.

Table 457 – IODReadReqHeader – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0009	RSP-, 0xDE, 0x81, 0x08, 0
BlockLength	!= 60	RSP-, 0xDE, 0x81, 0x08, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDE, 0x81, 0x08, 2
BlockVersionLow	!= 0x00	RSP-, 0xDE, 0x81, 0x08, 3
SeqNumber	Not checked	n. a.
ARUUID	Case RPCOperationNmb == 2 (Read): != ARBlockReq.ARUUID	RMPM: AR UUID unknown
ARUUID	Case RPCOperationNmb == 5 (Read Implicit): != NIL	RSP-, 0xDE, 0x81, 0x08, 5
API	> 0x0 AND not supported in GSDML	RSP-, 0xDE, 0x80, 0xB4, 6
SlotNumber	!= 0x0 to 0x7FFF OR not supported in GSDML	RSP-, 0xDE, 0x80, 0xB2, 7
SubslotNumber	!= 0x0 to 0x8FFF OR not supported in GSDML	RSP-, 0xDE, 0x80, 0xB2, 8
Padding	!= 0x0000	RSP-, 0xDE, 0x80, 0xB7, 9
Index	Not supported by application	RSP-, 0xDE, 0x80, 0xB0, 10
RecordDataLength	Not consistent with ArgsMaximum	RSP-, 0xDE, 0x81, 0x08, 11
TargetARUUID	Case RPCOperationNmb == 2 (Read): TargetARUUID != NIL	RSP-, 0xDE, 0x81, 0x08, 12
RWPadding	Not checked	n. a.

6.2.16.14.3 RecordDataReadQuery

This field shall be checked according to Table 458.

Table 458 – RecordDataReadQuery – request check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x0500 OR not supported in GSDML	RMPM: Unknown blocks
BlockLength	< 0x3 OR not consistent with ArgsLength	RSP-, 0xDE, 0x81, 0x08, 1
BlockVersionHigh	!= 0x01	RSP-, 0xDE, 0x81, 0x08, 2
BlockVersionLow	!= 0x00	RSP-, 0xDE, 0x81, 0x08, 3
Data	Not checked	n. a.

6.2.16.15 IODReadRes

6.2.16.15.1 ArgsLength

This field shall be checked according to Table 459.

Table 459 – ArgsLength check

	Parameter	Checking rules	Behaviour on match
NDRDataResponse	ArgsLength	!= IODReadResHeader.BlockLength + 4 + IODReadResHeader.RecordDataLength	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
		General: If one OR more unkonwn blocks are contained in the response	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
IODReadRes			

6.2.16.15.2 IODReadResHeader

This field shall be checked according to Table 460.

Table 460 – IODReadResHeader – response check

Parameter	Checking rules	Behaviour on match
BlockType	!= 0x8009	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockLength	!= 60	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionHigh	!= 0x01	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
BlockVersionLow	!= 0x00	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SeqNumber	Not checked	n. a.
ARUUID	!= IODReadReqHeader.ARUUID	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
API	!= IODReadReqHeader.API	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SlotNumber	!= IODReadReqHeader.SlotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
SubslotNumber	!= IODReadReqHeader.SubslotNumber	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Padding	!= 0x0000	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
Index	!= IODReadReqHeader.Index	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
RecordDataLength	Not consistent with ArgsLength	ERR-RTA, RTA_ERR_CLS_PROTOCOL, RTA_ERR_ABORT
AdditionalValue1	Not checked	n. a.
AdditionalValue2	Not checked	n. a.
RWPadding	Not checked	n. a.

6.2.16.15.3 RecordDataRead

This field shall be not checked.

6.3 FAL protocol state machines

6.3.1 Overall structure

6.3.1.1 Overview

The FAL protocol state machine structure is as defined in Figure 52. The general structure is according to the IEC 61158–6 series protocol machine model.

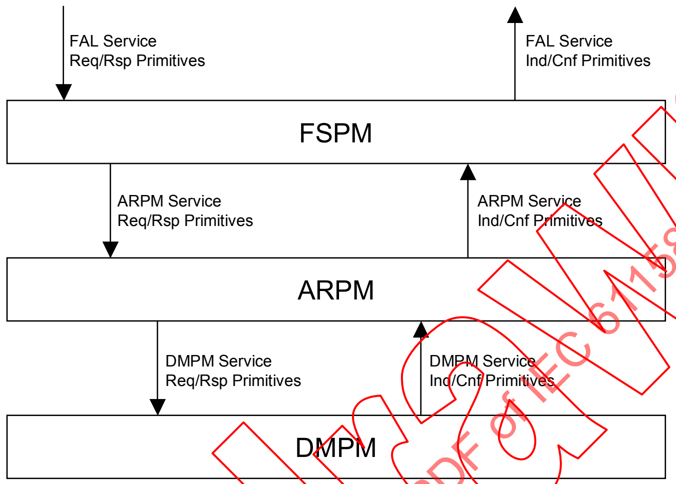

Figure 52 – Relationship among Protocol Machines

The behavior of the FAL is specified by three integrated protocol machines. The FSPM is the service interface between the FAL services that are part of the FAL Class specification and the particular AREP.

The FAL provides two kinds of protocol machine architectures, one for IO controller and one for IO devices. The architectures are specified in 5.3.3.2.1 and 5.3.3.3.

The FSPM is responsible for the following activities:

- To accept service primitives from the FAL service user and convert them into FAL internal primitives.
- To select the ARPM state machine based on the implicit addressing mechanism and send FAL internal primitives with the service parameters to the ARPM.
- To accept FAL internal primitives from the ARPM and convert them into service primitives for the FAL service user.
- To deliver the FAL service primitives to the FAL user.

The ARPM specifies the conveyance type for the application relation.

The DMPM specifies the mapping to the Data Link Layer. The DMPM defines therefore two protocol machines, the LMPM and the MAC protocol machines.

6.3.1.2 Fieldbus Service Protocol Machines (FSPM)

The FSPM State Machines co-ordinate the underlying state machines used for processing of the various services and application relations.

The FSPM basically is a mapping protocol machine. The main task is to pass the service to the protocol machine responsible for that service and forward confirmations and responses to the user. In addition a basic redundancy control scheme is included that allow to collaborate two AR into a single entity with higher availability.

For this task, the following support shall be offered by an FSPM:

- Redundant communication interface (RedCom) to a peer FSPM (only for redundant devices)
- Coordination of the Primary/Backup APDU_Status(information)
- Issuing switchover Procedures (signal Primary failures in case of an IO-Device)
- Collaboration of the set and get data services of the redundant AR

NOTE A switchover will be initiated in case of a communication problem (indicated by CPM via a NoData) if a switchover can be completed before Data hold time expires the system will continue otherwise the consumer will use Substitute Data for further processing.

6.3.1.3 IO controller to IO device (IO AR)

The PPM, CPM, ALPMI, ALPMR, APMR, APMS, CMDEV, CMCTL, RMPM State Machines are responsible for the cyclic, acyclic and alarm data transfer between an IO controller and an IO device.

6.3.1.4 IO supervisor to IO device (Supervisor AR)

The PPM, CPM, ALPMI, ALPMR, APMR, APMS, CMDEV, CMCTL, RMPM State Machines are responsible for the cyclic, acyclic and alarm data transfer between an IO supervisor and an IO device.

NOTE Within this application relationship, the PPM, CPM, APMR, and APMS protocol machines convey their APDUs directly over RPC and not over the LMPM.

6.3.1.5 DLL Mapping Protocol Machines (DMPM)

The DL Mapping Protocol Machines (DMPM) connects the other State machines and Layer 2. DMPM provides the coordination of all state machines concerning the configuration and error handling of the Data Link Layer Usage. The functions are mapped by the DMPM to the DLL services of Layer 2. The DMPM generates the necessary Layer 2 parameters of the service, receives the confirmations and indications from Layer 2 and passes them to the appropriate DMPM-User.

The DMPM is directly put upon the DLL User – DLL interface as described in Data Link Layer Service Definitions (Refer to Enhanced Internal Sublayer Services IEEE 802.3).

The DMPM uses for the mapping of the DMPM functions on the Layer 2 the following services:

- MA_UNITDATA.req
- MA_UNITDATA.ind

For the purpose of this specification there is an explicit confirmation for the MA_UNITDATA request service primitive indicating that the DLPDU has been transmitted.

6.4 AP-Context state machine

There is no AP-Context State Machine defined for this Protocol.

6.5 FAL service protocol machines (FSPMs)

6.5.1 Overview

This type specifies one FSPM state machine to transfer the FAL user service primitives into FAL internal service primitives and vice versa. There are two kinds of FSPM defined, one for the IO device (FSPMDEV) and one for the IO controller (FSPMCTL).

6.5.2 FSPMDEV

6.5.2.1 Overview

The FSPMDEV is responsible for the startup of the underlying protocol machines. They shall be started in the following order:

1) LMPM
2) LLDP
3) DHCP/DCP
4) RMPM (part of address resolution, the RMPM starts all other machines-IP, UDP, RPC, RM (other parts), CMDEV, PPM, CPM ...)

6.5.2.2 Primitive definitions

6.5.2.2.1 Primitives exchanged between FSPMDEV and AP-Context

Table 461 shows the service primitives including their associated parameters issued by the AP-Context (FAL user) and received by the FSPMDEV with the mapping to underlying services.

Table 461 – Primitives issued by AP-Context (FAL user) to FSPMDEV

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read.rsp(+)	FAL User	AREP Seq Number Add Data 1 Add Data 2 Length Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RSP+
Read.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_RSP-
Read Input Data.rsp(+)	FAL User	AREP Seq Number Length Length Data Length IOCS Length IOPS IOCS IOPS Subslot Input Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_INPD_RSP+
Read Input Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_INPD_RSP-
Read Output Data.rsp(+)	FAL User	AREP Seq Number Length Length IOCS Length IOPS Length Output Data IOCS IOPS Output Data Substitute Mode Substitute Active Flag Output Substitute Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_OUTPD_RSP+

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read Output Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_OUTPD_RSP-
Read Logbook.rsp(+)	FAL User	AREP Seq Number Length Current Local Time Stamp Number Of Log Entries List of Entries Local Time Stamp AR UUID PNIO Status Entry Detail	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_LOG_RSP+
Read Logbook.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_LOG_RSP-
Read Device Diagnosis.rsp(+)	FAL User	AREP Seq Number Length List of Diagnosis Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_DIAG_RSP+
Read Device Diagnosis.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_DIAG_RSP-
Read Expected Identification.rsp(+)	FAL User	AREP Seq Number Length List of Slots	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_EID_RSP+
Read Expected Identification.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_EID_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read Real Identification.rsp(+)	FAL User	AREP Seq Number Length Number of Slots List of Slots	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RID_RSP+
Read Real Identification.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_RID_RSP-
Read Identification Difference.rsp(+)	FAL User	AREP Seq Number Length Number of APIs List of APIs	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_IDDIFF_RSP+
Read Identification Difference.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_IDDIFF_RSP-
Read Real Port Data.rsp(+)	FAL User	AREP Seq Number Length Real List of Ports	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RPD_RSP+
Read Real Port Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_RPD_RSP-
Read Expected Port Data.rsp(+)	FAL User	AREP Seq Number Length Expected List of Ports	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_EPD_RSP+
Read Expected Port Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_EPD_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read Adjusted Port Data.rsp(+)	FAL User	AREP Seq Number Length Adjusted List of Ports	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_APD_RSP+
Read Adjusted Port Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_APD_RSP-
Read IR Data.rsp(+)	FAL User	AREP Seq Number Length IR Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_IRD_RSP+
Read IR Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_IRD_RSP-
Read Real Sync Data.rsp(+)	FAL User	AREP Seq Number Length Sync Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_RSYN_RSP+
Read Real Sync Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_RSYN_RSP-
Read Expected Sync Data.rsp(+)	FAL User	AREP Seq Number Length Sync Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_ESYN_RSP+
Read Expected Sync Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_ESYN_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Read PDev Data.rsp(+)	FAL User	AREP Seq Number Real List of Ports Expected List of Ports Adjusted List of Ports IR Data Sync Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_PDEV_RSP+
Read PDev Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_PDEV_RSP-
Read AR Data.rsp(+)	FAL User	AREP Seq Number Length Number Of ARs List of ARs	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_ARD_RSP+
Read AR Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_ARD_RSP-
Read IsoM Data.rsp(+)	FAL User	AREP Seq Number Length IsoM Data	CM_Read.rsp(+)(AREP, Seq Number, AddData1, AddData2, Length, Data)	MAP_READ_ISOM_RSP+
Read IsoM Data.rsp(-)	FAL User	AREP Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Read.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_READ_ISOM_RSP-
Write.rsp(+)	FAL User	AREP Multiple Seq Number Add Data 1 Add Data 2	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_RSP+

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Write.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-)(AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_RSP-
Write Expected Port Data.rsp(+)	FAL User	AREP Multiple Seq Number	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_EPD_RSP+
Write Expected Port Data.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-)(AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_EPD_RSP-
Write Adjusted Port Data.rsp(+)	FAL User	AREP Multiple Seq Number	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_APD_RSP+
Write Adjusted Port Data.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-)(AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_APD_RSP-
Write IR Data.rsp(+)	FAL User	AREP Multiple Seq Number	CM_Write.rsp(+) (AREP, Multiple, Seq Number, AddData1, AddData2)	MAP_WRITE_EPD_RSP+
Write IR Data.rsp(-)	FAL User	AREP Multiple Seq Number Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Write.rsp(-)(AREP, Multiple, SeqNumber, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_WRITE_EPD_RSP-
Set Input.req	FAL User	AREP CREP Slot Number Subslot Number IOPS Subslot Input Data	PPM_Set_Prov_Data.req (CREP,Data)	MAP_SIN_REQ
Set Output IOCS.req	FAL User	AREP CREP Slot Number Subslot Number IOCS	PPM_Set_Prov_Data.req (CREP,Data)	MAP_OIOCS_REQ

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Set Input APDU Data Status.req	FAL User	AREP DataValid Flag ARState Flag ProviderState Flag ProblemIndicator Flag	PPM_Set_Prov_Status.req (CREP,D_Status)	MAP_SAS_REQ
Get Input IOCS.req	FAL USER	AREP CREP Slot Number Subslot Number	CPM_Get_Cons_Status.req (CREP)	MAP_GINIOCS_REQ
Get Output.req	FAL User	AREP CREP Slot Number Subslot Number	CPM_Get_Cons_Data.req (CREP), CPM_Get_Cons_Status.req (CREP)	CREP=CREP
Alarm Notification.req	FAL User	AREP API Alarm Priority Alarm Type Slot Number Subslot Number Alarm Specifier Module Ident Number Submodule Ident Number Alarm Item	ALPMI_Alarm_Notification.req(CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, Module_Ident_Number, Submodule_Ident_Number, Alarm_User_Data_Structure_Identifier, Alarm_User_Data)	MAP_AN_REQ
Alarm Ack.req	FAL User	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier	ALPMR_Alarm_Ack.req(CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_AA_REQ
Connect.rsp(+)	FAL User	AREP AR Response Block List of IO CR Response Blocks Alarm CR Response Block Module Diff Block	CM_Connect.rsp(+)(AREP, ARBlockRes, ListOfIOCRBlockRes, AlarmCRBlockRes, ModuleDiffBlock)	MAP_CON_RSP+
Connect.rsp(-)	FAL User	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Connect.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_CON_RSP-
Release.rsp(+)	FAL User	AREP Session Key	CM_Release.rsp(+) (AREP, ControlBlock)	MAP_REL_RSP+
Release.rsp(-)	FAL User	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_Release.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_REL_RSP-

Primitive name	Source	Associated parameters	Primitive mapped to	Parameter mapped to
Abort.req	FAL User	AREP	CM_Abort.req (AREP)	AREP=AREP
End Of Parameter.rsp(+)	FAL User	AREP Session Key Alarm Sequence Number	CM_DControl.rsp(+) (AREP, ControlBlock)	MAP_EOP_RSP+
End Of Parameter.rsp(-)	FAL User	AREP Error Decode Error Code 1 Error Code 2 Add Data 1 Add Data 2	CM_DControl.rsp(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_EOP_RSP-
Application ready.req	FAL User	AREP Session Key Alarm Sequence Number Module Diff Block Control Block Properties	CM_CControl.req(AREP, ControlBlock, ModuleDiffBlock)	MAP_AREADY_REQ
local action			CM_Init.req	None
local action			IFW_Schedule_add.req(Port, Sched_list)	None
local action			IFW_Schedule_remove.req(Port)	None
local action			IFW_SetFWState.req(Port, FWState)	None
local action			IFW_SetLEState.req(Port, LEState)	None
local action			TMM_Set_Schedule_Item.req(Port, Sched_list)	None
local action			TMM_MasterAdd.req (SA, ProtVar, MS_list, MS_status)	None
local action			TMM_MasterRem.req (SA, ProtVar, MS_list, MS_status)	None
local action			RCTL_Stop.req	None
local action			SYN_Delay_Req.req (Port, DA, SA, ADQ_list, ADQ_status)	None
local action			SYN_Delay_Req.req (Port, DA, SA, ADQ_list, ADQ_status)	None

Table 462 shows the service primitives including their associated parameters issued by the FSPMDEV and received by the AP-Context (FAL user).

Table 462 – Primitives issued by FSPMDEV to AP-Context (FAL user)

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IND	Read.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Index Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_INPD_IND	Read Input Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_OUTPD_IND	Read Output Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_LOG_IND	Read Logbook.ind	FSPM	AREP Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_DIAG_IND	Read Device Diagnosis.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Diagnosis Item Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_EID_IND	Read Expected Identification.ind	FSPM	AREP Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_RID_IND	Read Real Identification.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IDDIFF_IND	Read Identification Difference.ind	FSPM	AREP Target ARUUID Seq Number Length

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_RPD_IND	Read Real Port Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_EPD_IND	Read Expected Port Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_APD_IND	Read Adjusted Port Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_IRD_IND	Read IR Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_RSYNC_IND	Read Real Sync Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_ESYNC_IND	Read Expected Sync Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_PDEV_IND	Read PDev Data.ind	FSPM	AREP Seq Number Length
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_ARD_IND	Read AR Data.ind	FSPM	AREP Target ARUUID Seq Number Length

Primitive mapped from	**Parameter mapped from**	**Primitive name**	**Source**	**Associated parameters**
CM_Read.ind(AREP, API, TargetARUUID, SlotNumber, SubslotNumber, Index, SeqNumber, Length)	MAP_READ_ISOM_IND	Read IsoM Data.ind	FSPM	AREP API Target ARUUID Slot Number Subslot Number Seq Number Length
CM_Write.ind(AREP, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_IND	Write.ind	FSPM	AREP API Slot Number Subslot Number Index Multiple Prm Flag Seq Number Length Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_OSUBD_IND	Write Output Substitute Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Substitute Mode Length Output Data Output Substitute Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_EPD_IND	Write Expected Port Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length Expected List of Ports
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_APD_IND	Write Adjusted Port Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length Adjusted List of Ports
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_IRD_IND	Write IR Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IR Data

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_ESYNC_IND	Write Sync Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length Sync Data
CM_Write.ind(AREP, API, SlotNumber, SubslotNumber, Index, Multiple, PrmFlag, SeqNumber, Length, Data)	MAP_WRITE_ISOM_IND	Write IsoM Data.ind	FSPM	AREP API Slot Number Subslot Number Multiple Prm Flag Seq Number Length IsoM Data
PPM_Set_Prov_Data.cnf(+) (CREP)	MAP_SIN_CNF	Set Input.cnf(+)	FSPM	AREP
PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_SIN_CNF	Set Input.cnf(-)	FSPM	AREP
PPM_Set_Prov_Status.cnf(+) (CREP)	MAP_SAS_CNF	Set Input APDU Data Status.cnf(+)	FSPM	AREP
PPM_Set_Prov_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_SAS_CNF	Set Input APDU Data Status.cnf(-)	FSPM	AREP
PPM_Set_Prov_Data.cnf(+) (CREP)	MAP_OIOCS_CNF	Set Output IOCS.cnf(+)	FSPM	AREP
PPM_Set_Prov_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_OIOCS_CNF	Set Output IOCS.cnf(-)	FSPM	AREP
CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)	MAP_GOUTD_CNF+	Get Ouput.cnf(+)	FSPM	AREP IOPS Subslot Output Data New Flag IOCS
CPM_Get_Cons_Data.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_GOUTD_CNF-	Get Ouput.cnf(-)	FSPM	AREP
CPM_Get_Cons_Status.cnf(+) (CREP, Status, RecvCounter)	MAP_GINIOCS_CNF+	Get Input IOCS.cnf(+)	FSPM	AREP IOCS
CPM_Get_Cons_Status.cnf(-) (CREP,ERRCLS,ERRCODE)	MAP_GINIOCS_CNF-	Get Input IOCS.cnf(-)	FSPM	AREP
CPM_Set_RedRole.cnf (CREP)	local action			
CM_Init.cnf	local action			
CM_Stop.ind	local action			

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_In_Data.ind(CREP)	MAP_INDATA_IND	New Output.ind	FSPM	AREP CREP Slot Number Subslot Number InData Flag
CM_New_Data.ind(CREP, APDU_Status)	MAP_NEWDATA_IND	New Output.ind	FSPM	AREP CREP Slot Number Subslot Number
CM_New_Data.ind(CREP, APDU_Status)	MAP_NEWSTATUS_IND	New Output APDU Data Status.ind	FSPM	AREP CREP Data Flag AR State Flag Provider State Flag Problem Indicator Flag
CPM_NoData.ind(CREP)	MAP_NODATA_IND	New Output.ind	FSPM	AREP CREP Slot Number Subslot Number Watchdog Flag
ALPMI_Alarm_Notification.cnf(+)(CREP)	MAP_AN_CNF+	Alarm notification.cnf(+)	FSPM	AREP
ALPMI_Alarm_Notification.cnf(-)(CREP)	MAP_AN_CNF-	Alarm notification.cnf(-)	FSPM	AREP Status
ALPMI_Alarm_Ack.ind(CREP, Alarm_Type, Slot_Number, Subslot_Number, Alarm_Specifier, Sequence_Number, PNIO_Status)	MAP_AA_IND	Alarm Ack.ind	FSPM	AREP API Alarm Type Slot Number Subslot Number Alarm Specifier
CM_Connect.ind(AREP, ARBlockReq, ListOfIOCRBlockReq, AlarmCRBlockReq, ListOfExpectedSubmoduleBlockReq, PrmServerBlock, ListOfMCRBlockReq)	MAP_CON_IND	Connect.ind	FSPM	AREP AR Parameter Block List of IO CR Parameter Blocks List of Expected Submodule Blocks Alarm CR Parameter Block Parameter Server Block List of Multicast CR Blocks
CM_Release.ind(AREP, ControlBlock)	MAP_REL_IND	Release.ind	FSPM	AREP Session Key
CM_Abort.ind	AREP	Abort.ind	FSPM	AREP
CM_Abort.cnf(AREP)	AREP=AREP	Abort.cnf	FSPM	AREP
CM_DControl.ind(AREP, ControlBlock)	MAP_EOP_IND	End Of Parameter.ind	FSPM	AREP Session Key Alarm Sequence Number
CM_CControl.cnf(+)(AREP, ControlBlock)	MAP_AREADY_CNF+	Application Ready.cnf(+)	FSPM	AREP Session Key Alarm Sequence Number

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
CM_CControl.cnf(-)(AREP, ErrorDecode, ErrorCode1, ErrorCode2, AddData1, AddData2)	MAP_AREADY_CNF-	Application Ready.cnf(-)	FSPM	AREP Error Decode Error Code 1 Error Code 2
LMPM_Time_Event.ind(CREP,Cycle)	MAP_SYNCH_IND	SYNCH Event.ind	FSPM	Slot Subslot Global Cycle Counter Phase Status
RCTL_Sync.ind (Port, DA, SA, CS_list, CS_status)	MAP_SYSTI_IND	Sync State Info.ind	FSPM	Real Sync Data Slot Number Subslot Number PTCP Subdomain ID IR Data ID Reserved Interval Begin Reserved Interval End PLL Window Sync Send Factor Send Clock Factor Sync Properties Role Stratum Sync Frame Address PTCP Timeout Factor Sync Error Status Slot Number Subslot Number
DiagnosisEvent.ind(AREP, CREP, API, Diagnosis Data)	MAP_DIAGEVT_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
ALPMI_Error.ind(CREP, ERRCLS, ERRCODE)	MAP_DIAGEVT_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
DCPUCR_Error.ind (CREP, ERRCLS, ERRCODE)	MAP_DIAGEVT_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
DCPMCR_Error.ind (CREP, ERRCLS, ERRCODE)	MAP_DIAGEVT_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
IFW_Error.ind(CREP, ERRCLS, ERRCODE)	MAP_DIAGEVT_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
IFW_Error.ind(CREP, ERRCLS, ERRCODE)	MAP_DIAGEVT_IND	Diagnosis Event.ind	FSPM	AREP CREP Alarm Item
TMM_MasterAdd.cnf(SA, PortVar, MS_list, MS_status)	local action			
TMM_MasterRem.cnf(SA, PortVar, MS_list, MS_status)	local action			
LMPM_Schedule_add.cnf (+) (CREP)	local action			

Primitive mapped from	Parameter mapped from	Primitive name	Source	Associated parameters
LMPM_Schedule_add.cnf (-) (CREP)	local action			
IFW_Schedule_add.cnf(Port)	local action			
IFW_Schedule_remove.cnf(Port)	local action			

6.5.2.2.2 Parameters of FSPMDEV primitives

The parameters used with the primitives exchanged between the FSPMDEV and the AP-Context are described in FAL Service Definition.

6.5.2.3 State machine description

FSPMDEV State Machine has got only one possible state: Run

6.5.2.4 FSPMDEV state table

In general, the FSPMDEV defines no state machine because all services are transferred to the underlying protocol machines.

However, in case of additional multicast communication between IO device the related state machine is defined in Table 463.

Table 463 – FSPMDEV protocol machine for multicast communication

#	Current State	Event /Condition =>Action	Next State
1	run	**Set Input.req(Slot Number, Subslot Number, IOPS, Input Data)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected => Data1 := Input Data, IOPS, CREP1 := CREP.IPPM Data2 := Input Data, IOPS CREP2 := CREP.MPPM PPM_Set_Prov_Data.req(CREP1, Data1) PPM_Set_Prov_Data.req(CREP2, Data2)	run
2	run	**Set Input.req(Slot Number, Subslot Number, IOPS, Input Data)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero => Data1 := IOPS CREP1 := CREP.IPPM Data2 := Input Data, IOPS CREP2 := CREP.MPPM PPM_Set_Prov_Data.req(CREP1, Data1) PPM_Set_Prov_Data.req(CREP2, Data2)	run
3	run	**Get Input IOCS.req(Slot Number, Subslot Number)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected => CREP := CREP.ICPM CPM_Get_Cons_Data.req(CREP)	run
4	run	**Get Input IOCS.req(Slot Number, Subslot Number)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero => Get Input IOCS.cnf(-)	run
5	run	**Set Output IOCS.req(Slot Number, Subslot Number, IOCS)** /submodule with M-Provider functionality && output functionality => CREP := CREP.OPPM PPM_Set_Prov_Data.req(CREP, IOCS)	run
6	run	**Set Output IOCS.req(Slot Number, Subslot Number, IOCS)** /submodule with M-Provider functionality && NOT (output functionality) => Set Output IOCS.cnf(-)	run
7	run	**Get Output.req(Slot Number, Subslot Number)** /submodule with M-Provider functionality && NOT(output functionality) => Get Output.req(-)	run

#	Current State	Event /Condition =>Action	Next State
8	run	**Get Output.req(Slot Number, Subslot Number)** /submodule with M-Provider functionality && output functionality => CREP := CREP.OCPM CPM_Get_Cons_Data.req(CREP)	run
9	run	**Set Input APDU Data Status.req(APDU data status flags)** /submodule with M-Provider functionality => CREP1 := CREP.IPPM CREP2 := CREP.MPPM D_Status := APDU data status flags PPM_Set_Prov_Status.req(CREP1, D_Status) PPM_Set_Prov_Status.req(CREP2, D_Status)	run
10	run	**PPM_Set_Prov_Data.cnf(+) (CREP)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input.cnf(+)	run
11	run	**PPM_Set_Prov_Data.cnf(+) (CREP)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run
12	run	**PPM_Set_Prov_Data.cnf(+) (CREP)** /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input.cnf(+)	run
13	run	**PPM_Set_Prov_Data.cnf(+) (CREP)** /submodule with M-Provider functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(+)	run
14	run	**PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input.cnf(-)	run
15	run	**PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run

#	Current State	Event /Condition =>Action	Next State
16	run	**PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input.cnf(-)	run
17	run	**PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(-)	run
18	run	**PPM_Set_Prov_Status.cnf(+) (CREP)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input APDU Data Status.cnf(+)	run
19	run	**PPM_Set_Prov_Status.cnf(+) (CREP)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run
20	run	**PPM_Set_Prov_Status.cnf(+) (CREP)** /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input APDU Data Status.cnf(+)	run
21	run	**PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Expected && CREP == CREP.IPPM => Set Input APDU Data Status.cnf(-)	run
22	run	**PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && SubmoduleProperties.ReduceInputSubmoduleDataLength == Zero && CREP == CREP.IPPM => ignore	run
23	run	**PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE** /submodule with M-Provider functionality && CREP == CREP.MPPM => Set Input APDU Data Status.cnf(-)	run
24	run	**CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)** /submodule with M-Provider functionality && CREP == CREP.OCPM => Get Output.cnf(+)(IOPS, Output Data, New Flag, IOCS)	run

#	Current State	Event /Condition =>Action	Next State
25	run	**CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)** /submodule with M-Provider functionality && CREP == CREP.ICPM => Get Input IOCS.cnf(IOCS)	run
26	run	**CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && CREP == CREP.OCPM => Get Output.cnf(-)	run
27	run	**CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Provider functionality && CREP == CREP.ICPM => Get Input IOCS.cnf(-)	run
28	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Provider functionality && CREP == CREP.OCPM && no change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
29	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Provider functionality && CREP == CREP.OCPM && change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
30	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Provider functionality && CREP == CREP.ICPM && no change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
31	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Provider functionality && CREP == CREP.ICPM && change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
32	run	**CPM_NoData.ind(CREP)** /submodule with M-Provider functionality && CREP == CREP.OCPM => Watchdog Flag := WATCHDOG_EXPIRED New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run

#	Current State	Event /Condition =>Action	Next State
33	run	**CPM_NoData.ind(CREP)** /submodule with M-Provider functionality && CREP == CREP.ICPM => Watchdog Flag := WATCHDOG_EXPIRED New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
34	run	**CPM_Start.ind(CREP)** /submodule with M-Provider functionality => InData Flag := INDATA New Output.ind(Slot Number, Subslot Number, InData Flag) New Output APDU Data Status.ind(AREP, CREP, APDU status flags)	run
35	run	**Set Input.req(Slot Number, Subslot Number, IOPS, Input Data)** /submodule with M-Consumer functionality && input functionality => Data := Input Data, IOPS CREP := CREP.IPPM PPM_Set_Prov_Data.req(CREP, Data)	run
36	run	**Set Input.req(Slot Number, Subslot Number, IOPS, Input Data)** /submodule with M-Consumer functionality && NOT (input functionality) => Set Input.cnf(-)	run
37	run	**Get Input IOCS.req(Slot Number, Subslot Number)** /submodule with M-Consumer functionality && input functionality => CREP := CREP.ICPM CPM_Get_Cons_Data.req(CREP)	run
38	run	**Get Input IOCS.req(Slot Number, Subslot Number)** /submodule with M-Consumer functionality && NOT (input functionality) => Get Input IOCS.cnf(-)	run
39	run	**Set Output IOCS.req(Slot Number, Subslot Number, IOCS)** /submodule with M-Consumer functionality => CREP := CREP.OPPM PPM_Set_Prov_Data.req(CREP, IOCS)	run
40	run	**Get Output.req(Slot Number, Subslot Number)** /submodule with M-Consumer functionality => CREP := CREP.MCPM CPM_Get_Cons_Data.req(CREP)	run
41	run	**Set Input APDU Data Status.req(AREP, CREP, APDU data status flags)** /submodule with M-Consumer functionality => CREP := CREP.IPPM D_Status := APDU data status flags PPM_Set_Prov_Status.req(CREP, D_Status)	run

#	Current State	Event /Condition =>Action	Next State
42	run	**PPM_Set_Prov_Data.cnf(+) (CREP)** /submodule with M-Consumer functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(+)	run
43	run	**PPM_Set_Prov_Data.cnf(+) (CREP)** /submodule with M-Consumer functionality && CREP == CREP.IPPM => Set Input.cnf(+)	run
44	run	**PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Consumer functionality && CREP == CREP.OPPM => Set Output IOCS.cnf(-)	run
45	run	**PPM_Set_Prov_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Consumer functionality && CREP == CREP.IPPM => Set Input.cnf(-)	run
46	run	**PPM_Set_Prov_Status.cnf(+) (CREP)** /submodule with M-Consumer functionality => Set Input APDU Data Status.req(+)	run
47	run	**PPM_Set_Prov_Status.cnf(-) (CREP, ERRCLS, ERRCODE** /submodule with M-Consumer functionality => Set Input APDU Data Status.req(-)	run
48	run	**CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)** /submodule with M-Consumer functionality && CREP := MCPM => Get Output.cnf(+)(IOPS, Output Data, New Flag)	run
49	run	**CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)** /submodule with M-Consumer functionality && CREP := OCPM => ignore	run
50	run	**CPM_Get_Cons_Data.cnf(+) (CREP, Data, New_Flag)** /submodule with M-Consumer functionality && CREP := ICPM => Get Input IOCS.cnf(IOCS)	run
51	run	**CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Consumer functionality && CREP := MCPM => Get Output.cnf(-)	run

#	Current State	Event /Condition =>Action	Next State
52	run	**CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Consumer functionality && CREP := OCPM => ignore	run
53	run	**CPM_Get_Cons_Data.cnf(-) (CREP, ERRCLS, ERRCODE)** /submodule with M-Consumer functionality && CREP := ICPM => Get Input IOCS.cnf(-)	run
54	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Consumer functionality && CREP == CREP.OCPM => ignore	run
55	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Consumer functionality && CREP == CREP.ICPM => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
56	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Consumer functionality && CREP == CREP.ICPM => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
57	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Consumer functionality && CREP == CREP.MCPM && no change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
58	run	**CPM_New_Cons_Data.ind(CREP,APDU_Status)** /submodule with M-Consumer functionality && CREP == CREP.MCPM && change in APDU status => New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag) New Output APDU Data Status.ind(APDU status flags)	run
59	run	**CPM_No_Data.ind(CREP)** /submodule with M-Consumer functionality => Watchdog Flag := WATCHDOG_EXPIRED New Output.ind(Slot Number, Subslot Number, Watchdog Flag, InData Flag)	run
60	run	**CPM_Start.ind(CREP)** /submodule with M-Consumer functionality => InData Flag := INDATA New Output.ind(Slot Number, Subslot Number, In Data Flag)	run

#	Current State	Event /Condition =>Action	Next State
61	run	**CM_Stop.ind(CREP)** /submodule with M-Consumer functionality && CREP == CREP.MCPM => For all submoduls of M-Consumer-CR do: M-Provider Communication Stopped.ind(Slot Number, Subslot Number)	run

6.5.2.5 Functions

Table 464 shows the functions defined for FSPMDEV, which are used by service primitives issued by the FAL user.

Table 464 – Functions used by AP-Context (FAL user) to FSPMDEV

Name	Function
MAP_READ_RSP+	map service parameter 1:1 (=)
MAP_READ_RSP-	map service parameter 1:1 (=)
MAP_READ_INPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Length Data, Length IOCS, Length IOPS, IOCS, IOPS, Subslot Input Data}
MAP_READ_INPD_RSP-	map service parameter 1:1 (=)
MAP_READ_OUTPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Length IOCS, Length IOPS, Length Output Data, IOCS, IOPS, Output Data, Substitute Mode, Substitute Active Flag, Output Substitute Data}
MAP_READ_OUTPD_RSP-	map service parameter 1:1 (=)
MAP_READ_LOG_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Current Local Time Stamp, Number of Log Entries, [List of Entries (Local Time Stamp, AR UUID, PNIO Status, Entry Detail)]*}
MAP_READ_LOG_RSP-	map service parameter 1:1 (=)
MAP_READ_DIAG_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Diagnosis Item=List of Diagnosis Data
MAP_READ_DIAG_RSP-	map service parameter 1:1 (=)

Name	Function
MAP_READ_EID_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={List of Slots}
MAP_READ_EID_RSP-	map service parameter 1:1 (=)
MAP_READ_RID_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Number of Slots, List of Slots}
MAP_READ_RID_RSP-	map service parameter 1:1 (=)
MAP_READ_IDDIFF_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Number of APIs, List of APIs}
MAP_READ_IDDIFF_RSP-	map service parameter 1:1 (=)
MAP_READ_RPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Real List of Ports}
MAP_READ_RPD_RSP-	map service parameter 1:1 (=)
MAP_READ_EPD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Expected List of Ports}
MAP_READ_EPD_RSP-	map service parameter 1:1 (=)
MAP_READ_APD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Adjusted List of Ports}
MAP_READ_APD_RSP-	map service parameter 1:1 (=)
MAP_READ_IRD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={{IR Data}
MAP_READ_IRD_RSP-	map service parameter 1:1 (=)

Name	Function
MAP_READ_RSYNC_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Sync Data}
MAP_READ_RSYNC_RSP-	map service parameter 1:1 (=)
MAP_READ_ESYNC_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Sync Data}
MAP_READ_ESYNC_RSP-	map service parameter 1:1 (=)
MAP_READ_PDEV_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data= {Real Port Data, Expected Port Data, IR Data, Real Sync Data, Expected Sync Data}
MAP_READ_PDEV_RSP-	map service parameter 1:1 (=)
MAP_READ_ARD_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={Number of ARs, List of ARs}
MAP_READ_ARD_RSP-	map service parameter 1:1 (=)
MAP_READ_ISOM_RSP+	AREP=AREP Seq Number=Seq Number Add Data 1=0 Add Data 2=0 Length=Length Data={IsoM Data}
MAP_READ_ISOM_RSP-	map service parameter 1:1 (=)
MAP_WRITE_RSP+	map service parameter 1:1 (=)
MAP_WRITE_RSP-	map service parameter 1:1 (=)
MAP_WRITE_EPD_RSP+	map FAL User service parameter 1:1 (=) set AddData1=0 set AddData2=0
MAP_WRITE_EPD_RSP-	map service parameter 1:1 (=)
MAP_WRITE_APD_RSP+	map FAL User service parameter 1:1 (=) set AddData1=0 set AddData2=0
MAP_WRITE_APD_RSP-	map service parameter 1:1 (=)
MAP_WRITE_ISOM_RSP+	map FAL User service parameter 1:1 (=) set AddData1=0 set AddData2=0
MAP_WRITE_ISOM_RSP-	map service parameter 1:1 (=)

Name	Function
MAP_CON_RSP+	AREP=AREP ARBlockRes=AR Response Block ListOfIOCRBlockRes=List of IO CR Response Blocks AlarmCRBlockRes=Alarm CR Response Block ModuleDiffBlock=Module Diff Block
MAP_CON_RSP-	map service parameter 1:1 (=)
MAP_REL_RSP+	AREP=AREP ReleaseBlock.SessionKey=SessionKey
MAP_REL_RSP-	map service parameter 1:1 (=)
MAP_AREADY_REQ	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number) ModuleDiffBlock=Module Diff Block
MAP_EOP_RSP+	AREP=AREP ControlBlockConnect.SessionKey=Session Key^(ControlBlockPlug.SessionKey=Session Key,ControlBlockPlug.AlarmSequenceNumber=Alarm Sequence Number)
MAP_EOP_RSP-	map service parameter 1:1 (=)
MAP_SIN_REQ	CREP=CREP Data={IOPS, Subslot Input Data}
MAP_GINIOCS_REQ	CREP=CREP
MAP_SAS_REQ	CREP=AREP.CREP D_Status={DataValid Flag, AR State Flag, ProviderState Flag, Problemindicator Flag}
MAP_OIOCS_REQ	CREP=CREP Data=IOCS
MAP_AN_REQ	map service parameter 1:1 (=)

Table 465 shows the functions defined for FSPMDEV, which are used by services primitives issued by the FSPMDEV.

Table 465 – Function used by FSPMDEV to AP-Context (FAL user)

Name	Function
MAP_READ_IND	if ((Index < =0x7FFF) OR (0xAFF0<=Index<=0xAFFF)) then {AREP=AREP API=API Target ARUUID = TargetARUUID Slot Number = SlotNumber Subslot Number =SubslotNumber Index =Index Seq Number =SeqNumber Length=Length} else map to special FAL user Read service primitive
MAP_READ_INPD_IND	if (Index =0x8028) then {AREP=AREP API=API Target ARUUID = TargetARUUID Slot Number = SlotNumber Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_OUTPD_IND	if (Index =0x8029) then {AREP=AREP API=API Target ARUUID = TargetARUUID Slot Number = SlotNumber Subslot Number =SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive
MAP_READ_LOG_IND	if (Index =0xF830) then {AREP=AREP Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

Name	Function

MAP_READ_DIAG_IND	if ((Index =0x800A)OR(Index =0x800B)OR(Index =0x800C)OR
	(Index =0xC00A)OR(Index =0xC00B)OR(Index =0xC00C)OR
	(Index =0xE00A)OR(Index =0xE00B)OR(Index =0xE00C)OR
	(Index =0xF00A)OR(Index =0xF00B)OR(Index =0xF00C))then
	{AREP=AREP
	if (TargetARUUID<>NULL) then Target ARUUID=TargetARUUID else Target ARUUID ="IGNORE"
	if (Index =0xE00A)OR(Index =0xE00B)OR(Index =0xE00C) then API = "IGNORE else API=API
	if ((Index =0xF00A)OR(Index =0xF00B)OR(Index =0xF00C) OR
	(Index =0xE00A)OR(Index =0xE00B)OR(Index =0xE00C)) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xF00A)OR(Index =0xF00B)OR(Index =0xF00C) OR
	(Index =0xC00A)OR(Index =0xC00B)OR(Index =0xC00C)OR
	(Index =0xE00A)OR(Index =0xE00B)OR(Index =0xE00C)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	if ((Index =0x800A)OR(Index =0xC00A) OR(Index =0xE00A)OR(Index =0xF00A) then Diagnosis Item = CHANNEL
	if ((Index =0x800B)OR(Index =0xC00B) OR(Index =0xE00B)OR(Index =0xE00B)OR(Index =0xF00B) then Diagnosis Item = MANUFACTURER_ERROR_APPEARS
	if ((Index =0x800C)OR(Index =0xC00C) OR(Index =0xE00C)OR(Index =0xF00C) then Diagnosis Item = MANUFACTURER_ALL
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_EID_IND	if ((Index =0x8000)OR(Index =0xC000)OR(Index =0xE000)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE000) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE000)OR(Index =0xC000)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_RID_IND	if ((Index =0x8001)OR(Index =0xC001)OR(Index =0xE001)OR(Index =0xF000)) then
	{AREP=AREP
	API=API
	if (Index =0xF001) then Target ARUUID = "IGNORE"
	else Target ARUUID = TargetARUUID
	if ((Index =0xE001)OR(Index =0xF001)) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE001)OR(Index =0xC001)OR(Index =0xF000)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

Name	Function
MAP_READ_IDDIFF_IND	if (Index =0xE002) then {AREP=AREP Target ARUUID = TargetARUUID Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

MAP_READ_RPD_IND	if ((Index =0x802A)OR(Index =0xC02A)OR(Index =0xE02A)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE02A) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE02A)OR(Index =0xC02A)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_EPD_IND	if ((Index =0x802B)OR(Index =0xC02B)OR(Index =0xE02B)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE02B) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE02B)OR(Index =0xC02B)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_APD_IND	if ((Index =0x802F)OR(Index =0xC02F)OR(Index =0xE02F)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE02F) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE02F)OR(Index =0xC02F)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_IRD_IND	if ((Index =0x802C)OR(Index =0xC02C)OR(Index =0xE02C)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE02C) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE02C)OR(Index =0xC02C)) then Subslot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

Name	Function

MAP_READ_RSYNC_IND	if ((Index =0x802D)OR(Index =0xC02D)OR(Index =0xE02D)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE02D) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE02D)OR(Index =0xC02D)) then Subslot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_ESYNC_IND	if ((Index =0x802E)OR(Index =0xC02E)OR(Index =0xE02E)) then
	{AREP=AREP
	Target ARUUID = TargetARUUID
	if (Index =0xE02E) then Slot Number = "IGNORE"
	else Slot Number =SlotNumber
	if ((Index =0xE02E)OR(Index =0xC02E)) then Subslot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_PDEV_IND	if (Index =0xF831) then {AREP=AREP Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

MAP_READ_ARD_IND	if ((Index =0x8001)OR(Index =0xC001)OR(Index =0xF020)OR(Index =0xF820)) then
	{AREP=AREP
	if (Index =0xF820) then API = "IGNORE"
	else API=API
	if ((Index =0xF820)OR(Index =0xF020)) then Target ARUUID = "IGNORE"
	else Target ARUUID = TargetARUUID
	Slot Number = "IGNORE"
	Sublot Number = "IGNORE"
	Seq Number =SeqNumber
	Length=Length}
	else map to other special FAL user Read service primitive

MAP_READ_ISOM_IND	if (Index =0x8030) then {AREP=AREP API=API Target ARUUID = TargetARUUID Slot Number = SlotNumber Sublot Number = SubslotNumber Seq Number =SeqNumber Length=Length} else map to other special FAL user Read service primitive

Name	Function
MAP_WRITE_IND	if ((Index < =0x7FFF) OR (0xAFF0<=Index<=0xAFFF)) then {AREP=AREP API=API Slot Number = SlotNumber Subslot Number =SubslotNumber Index =Index Multiple = Multiple Seq Number =SeqNumber Length=Length Data=Data} else map to special FAL user Write service primitive
MAP_WRITE_OSUBD_IND	if ((Index =0x801E)) then {AREP=AREP API=API Slot Number = SlotNumber Subslot Number =SubslotNumber Index =Index Multiple = Multiple Seq Number =SeqNumber Length=Length {Substitute Mode, Length Output Data, Output Substitute Data} = Data}

MAP_WRITE_EPD_IND	if ((Index =0x802B)OR(Index =0xC02B)OR(Index =0xE02B)) then
	{AREP=AREP
	API=API
	if (Index =0xE02B) then Slot Number = "IGNORE"
	else Slot Number = SlotNumber
	if ((Index =0xE02B)OR(Index =0xC02B)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Index =Index
	Multiple = Multiple
	Seq Number =SeqNumber
	Length=Length
	{Expected List of Ports} = Data}
	else map to special FAL user Write service primitive

MAP_WRITE_APD_IND	if ((Index =0x802F)OR(Index =0xC02F)OR(Index =0xE02F)) then
	{AREP=AREP
	API=API
	if (Index =0xE02F) then Slot Number = "IGNORE"
	else Slot Number = SlotNumber
	if ((Index =0xE02F)OR(Index =0xC02F)) then Sublot Number = "IGNORE"
	else Subslot Number =SubslotNumber
	Index =Index
	Multiple = Multiple
	Seq Number =SeqNumber
	Length=Length
	{Adjusted List of Ports} = Data}
	else map to special FAL user Write service primitive