

INTERNATIONAL STANDARD

IEC
61158-6

Second edition
2000-01

**Digital data communications for
measurement and control –
Fieldbus for use in industrial control systems –**

**Part 6:
Application Layer protocol specification**



Reference number
IEC 61158-6:2000(E)

Numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series.

Consolidated publications

Consolidated versions of some IEC publications including amendments are available. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Validity of this publication

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology.

Information relating to the date of the reconfirmation of the publication is available in the IEC catalogue.

Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is to be found at the following IEC sources:

- **IEC web site***
- **Catalogue of IEC publications**
Published yearly with regular updates
(On-line catalogue)*
- **IEC Bulletin**
Available both at the IEC web site* and as a printed periodical

Terminology, graphical and letter symbols

For general terminology, readers are referred to IEC 60050: *International Electrotechnical Vocabulary* (IEV).

For graphical symbols, and letter symbols and signs approved by the IEC for general use, readers are referred to publications IEC 60027: *Letter symbols to be used in electrical technology*, IEC 60417: *Graphical symbols for use on equipment. Index, survey and compilation of the single sheets* and IEC 60617: *Graphical symbols for diagrams*.

* See web site address on title page.

INTERNATIONAL STANDARD

IEC
61158-6

Second edition
2000-01

Digital data communications for measurement and control – Fieldbus for use in industrial control systems –

Part 6: Application Layer protocol specification

© IEC 2000 – Copyright - all rights reserved

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

International Electrotechnical Commission
Telefax: +41 22 919 0300

3, rue de Varembé Geneva, Switzerland
e-mail: inmail@iec.ch IEC web site <http://www.iec.ch>



Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE **XH**

For price, see current catalogue

CONTENTS

INTRODUCTION	22
1 Scope.....	23
2 Normative references	24
3 Definitions	25
3.1 Definitions from other ISO/IEC Standards	25
3.1.1 Definitions from ISO/IEC 7498-1	25
3.1.2 Definitions from ISO/IEC 8822	25
3.1.3 Definitions from ISO/IEC 9545	25
3.1.4 Definitions from ISO/IEC 8824	25
3.1.5 Definitions from ISO/IEC 8825	25
3.2 Definitions from IEC 61158-5	25
3.3 Other definitions	26
3.3.1 Type 1 Definitions	26
3.3.1.1 called.....	26
3.3.1.2 calling.....	26
3.3.1.3 interoperability	26
3.3.1.4 management information.....	26
3.3.1.5 receiving.....	26
3.3.1.6 resource	26
3.3.1.7 sending.....	26
3.3.2 Type 2 Definitions	26
3.3.2.1 Allocate	26
3.3.2.2 Application.....	26
3.3.2.3 Application objects.....	26
3.3.2.4 Attribute.....	26
3.3.2.5 Behaviour	27
3.3.2.6 Class.....	27
3.3.2.7 Class attributes.....	27
3.3.2.8 Class code.....	27
3.3.2.9 Class specific service	27
3.3.2.10 Client.....	27
3.3.2.11 Connection	27
3.3.2.12 Connection ID (CID).....	27
3.3.2.13 Connection path.....	27
3.3.2.14 Connection point.....	27
3.3.2.15 Consume	27
3.3.2.16 Consumer.....	27
3.3.2.17 Consuming application.....	27
3.3.2.18 Cyclic	27
3.3.2.19 Device	27
3.3.2.20 Device profile.....	28
3.3.2.21 End node	28
3.3.2.22 End point	28
3.3.2.23 Error.....	28

3.3.2.24	Frame.....	28
3.3.2.25	Instance.....	28
3.3.2.26	Instance attributes	28
3.3.2.27	Instantiated	28
3.3.2.28	Keeper.....	28
3.3.2.29	Little endian	28
3.3.2.30	Lpacket.....	28
3.3.2.31	Member	28
3.3.2.32	Message router	28
3.3.2.33	Multicast connection	28
3.3.2.34	Network	28
3.3.2.35	Object.....	28
3.3.2.36	Object specific service	29
3.3.2.37	Originator	29
3.3.2.38	Point-to-point connection	29
3.3.2.39	Produce	29
3.3.2.40	Producer.....	29
3.3.2.41	Server	29
3.3.2.42	Service	29
3.3.2.43	Target.....	29
3.3.2.44	Temporary node.....	29
3.3.2.45	Transaction id	29
3.3.2.46	Unconnected message manager (UCMM)	29
3.3.2.47	Unconnected service.....	29
3.3.3	Type 3 Definitions	29
3.3.3.1	Access Protection	29
3.3.3.2	Address-assignment-table.....	29
3.3.3.3	Channel.....	29
3.3.3.4	Channel related Diagnosis	30
3.3.3.5	Configuration check	30
3.3.3.6	Configuration fault	30
3.3.3.7	Configuration Identifier.....	30
3.3.3.8	Configuration Identifier related Diagnosis	30
3.3.3.9	Control Commands	30
3.3.3.10	Data consistency	30
3.3.3.11	Default address	30
3.3.3.12	Device related diagnosis	30
3.3.3.13	Diagnosis information	30
3.3.3.14	Diagnosis information collection	30
3.3.3.15	DP-Master (Class 1)	30
3.3.3.16	DP-Master (Class 2)	30
3.3.3.17	DP-Slave	30
3.3.3.18	Freeze	30
3.3.3.19	Group	30
3.3.3.20	I/O data	31
3.3.3.21	Ident Number	31
3.3.3.22	Index	31
3.3.3.23	Master parameter set.....	31

3.3.3.24	Module	31
3.3.3.25	Process data	31
3.3.3.26	Real Configuration	31
3.3.3.27	Slot	31
3.3.3.28	Sync	31
3.3.4	Type 4 Definitions	31
3.3.5	Type 5 Definitions	31
3.3.6	Type 6 Definitions	31
3.3.7	Type 7 Definitions	31
3.3.8	Type 8 Definitions	31
3.4	Abbreviations and symbols	31
3.4.1	Type 1 Abbreviations and symbols	31
3.4.2	Type 2 Abbreviations and symbols	32
3.4.3	Type 3 Abbreviations and symbols	32
3.4.4	Type 4 Abbreviations and symbols	33
3.4.5	Type 5 Abbreviations and symbols	33
3.4.6	Type 6 Abbreviations and symbols	33
3.4.7	Type 7 Abbreviations and symbols	33
3.4.8	Type 8 Abbreviations and symbols	34
3.5	Conventions	34
3.5.1	Conventions for Type 1	34
3.5.1.1	Conventions for Class Definitions	34
3.5.1.2	Abstract Syntax Conventions	34
3.5.2	Conventions for Type 2	34
3.5.2.1	Attribute specification	34
3.5.2.2	Common Services	35
3.5.3	Conventions for Type 3	38
3.5.3.1	Abstract Syntax Conventions	38
3.5.3.2	Convention for the Encoding of Reserved Bits and Octets	38
3.5.3.3	Conventions for the Common Coding Structure of specific Field Octets	38
3.5.4	Conventions for Type 4	39
3.5.5	Conventions for Type 5	39
3.5.6	Conventions for Type 6	39
3.5.7	Conventions for Type 7	39
3.5.8	Conventions for Type 8	39
3.6	Conventions used in State Machines	39
3.6.1	Conventions for Type 1	39
3.6.2	Conventions for Type 2	40
3.6.2.1	State Machine Conventions	40
3.6.3	Conventions for Type 3	42
3.6.4	Conventions for Type 4	44
3.6.5	Conventions for Type 5	44
3.6.6	Conventions for Type 6	44
3.6.7	Conventions for Type 7	44
3.6.7.1	Evaluation networks	45
3.6.7.2	Simple actions	45
3.6.7.3	Conditions	45
3.6.8	Conventions for Type 8	45

4	Type 1	46
4.1	FAL Syntax Description	46
4.1.1	FAL-AR PDU Abstract Syntax 1	46
4.1.1.1	Confirmed Send Service	46
4.1.1.2	Unconfirmed Send Service	46
4.1.1.3	Unconfirmed Acknowledged Send Service	46
4.1.1.4	Idle Send Service	47
4.1.1.5	AR-XON-OFF Send Service	47
4.1.1.6	Establish Service	47
4.1.2	FAL-AR PDU Abstract Syntax 2	47
4.1.2.1	Confirmed Send Service	47
4.1.2.2	Unconfirmed Send Service	48
4.1.2.3	Unconfirmed Acknowledged Send Service	48
4.1.2.4	Idle Send Service	48
4.1.2.5	AR-XON-OFF Send Service	48
4.1.2.6	Establish Service	48
4.1.2.7	MaxOSCC	49
4.1.2.8	MaxOSCS	49
4.1.2.9	MaxUCSC	49
4.1.2.10	MaxUCSS	49
4.1.2.11	XON_OFF	49
4.1.2.12	CIU	49
4.1.3	Abstract Syntax of PDUBody	49
4.1.3.1	Abort Service	49
4.1.3.2	InvokeID	49
4.1.3.3	ConfirmedServiceRequest	50
4.1.3.4	ConfirmedServiceResponse	51
4.1.3.5	ConfirmedServiceError	52
4.1.3.6	Error Type	52
4.1.3.7	Error Class	53
4.1.3.8	Unconfirmed PDUs	54
4.1.3.9	Management ASE	54
4.1.3.10	Application Process ASE	56
4.1.3.11	Load Region ASE	57
4.1.3.12	Function Invocation ASE	59
4.1.3.13	Variable Access ASE	60
4.1.3.14	Event Management ASE	63
4.1.3.15	Type Definitions	66
4.1.4	Data Types	82
4.1.4.1	Notation for the Boolean Type	82
4.1.4.2	Notation for the Integer Type	82
4.1.4.3	Notation for the Unsigned Type	82
4.1.4.4	Notation for the Floating Point Type	82
4.1.4.5	Notation for the BitString Type	83
4.1.4.6	Notation for the OctetString Type	83
4.1.4.7	Notation for VisibleString Type	83
4.1.4.8	Notation for the UNICODEString Type	83
4.1.4.9	Notation for the FieldbusTime Type	83

4.1.4.10	Notation for the Universal Time Type	83
4.1.4.11	Notation for Binary Time Type	83
4.1.4.12	Notation for BCD Type	83
4.1.4.13	Notation for Compact Boolean Array Type	83
4.1.4.14	Notation for Compact BCD Array Type	83
4.1.4.15	Notation for BinaryDate Type	83
4.1.4.16	Notation for TimeOfDay Type	83
4.1.4.17	Notation for TimeDifference Type	83
4.1.4.18	Notation for TimeValue Type	83
4.1.5	Reason Codes	84
4.1.5.1	Introduction	84
4.2	Transfer Syntaxes	85
4.2.1	Transfer Syntax 1	85
4.2.1.1	FAL Header	85
4.2.1.2	Traditional Encoding Rule (TER)	86
4.2.1.3	Compact Encoding Rule (CER)	94
4.3	FAL Protocol State Machines Structure	109
4.4	AP Context State Machine	110
4.4.1	Primitive Definitions	110
4.4.1.1	Primitives Exchanged between FAL-User and AP-Context	110
4.4.2	State Machine Description	110
4.4.3	AP-AP Context Initiation State Transitions	111
4.4.4	Functions	122
4.5	FAL Service Protocol Machine (FSPM)	125
4.5.1	Primitive Definitions	125
4.5.1.1	Primitives Exchanged between AP_Context and FSPM	125
4.5.1.2	Parameters of AP_Context/FSPM Primitives	126
4.5.2	FSPM State Tables	127
4.5.2.1	Functions	130
4.6	Application Relationship Protocol Machines (ARPMs)	131
4.6.1	Queued Usertriggered Unidirectional (QUU) ARPM	131
4.6.1.1	Primitive Definitions	131
4.6.1.2	Parameters of FSPM/ARPM Primitives	131
4.6.1.3	QUU ARPM State Machine	134
4.6.2	Queued Usertriggered Bidirectional-Connection Oriented (QUB-CO) ARPM	137
4.6.2.1	Primitive Definitions	137
4.6.2.2	DLL Mapping of QUB AREP Class	137
4.6.2.3	QUB AREP State Machine	141
4.6.3	Queued Usertriggered Bidirectional-Connectionless (QUB-CI) ARPM	150
4.6.3.1	Primitive Definitions	150
4.6.3.2	DLL Mapping of QUB-CL AREP Class	150
4.6.3.3	QUB-CL ARPM State Machine	152
4.6.4	Queued Usertriggered Bidirectional-Segmentation (QUB-Seg) ARPM	157
4.6.4.1	Primitive Definitions	157
4.6.4.2	Parameters of FSPM/ARPM Primitives	157
4.6.4.3	QUB-Seg ARPM State Machine	161
4.6.5	Queued Usertriggered Bidirectional-Flow Control (QUB-FC) ARPM	175
4.6.5.1	QUB-FC Primitive Definitions	175

4.6.5.2	Parameters of FSPM/ARPM Primitives	176
4.6.5.3	QUB-FC ARPM State Machine	179
4.6.6	Buffered Usertriggered Bidirectional (BUB) ARPM	201
4.6.6.1	Primitive Definitions	201
4.6.6.2	DLL Mapping of BUB AREP Class	202
4.6.6.3	BUB ARPM State Machine	205
4.6.7	Buffered Networkscheduled Bidirectional (BNB) ARPM	216
4.6.7.1	Primitive Definitions	216
4.6.7.2	Primitives issued by ARPM to FSPMParameters of FSPM/ARPM Primitives	216
4.6.7.3	DLL Mapping of BNB AREP Class	217
4.6.7.4	BNB ARPM state machine	220
4.6.8	Buffered Networkscheduled Unidirectional (BNU) ARPM	240
4.6.8.1	Primitive Definitions	240
4.6.8.2	DLL Mapping of BNU AREP Class	241
4.6.8.3	BNU AREP State Machine	244
4.6.9	Buffered Networkscheduled Unidirectional-MP (BNU-MP) ARPM	251
4.6.9.1	Primitive Definitions	251
4.6.9.2	DLL Mapping of BNU-MP AREP Class	252
4.6.9.3	BNU-MP ARPM State Machine	255
4.7	DLL Mapping Protocol Machine (DMPM)	263
4.7.1	Primitive Definitions	263
4.7.1.1	Primitives Exchanged between DMPM and ARPM	263
4.7.1.2	Parameters of ARPM/DMPM Primitives	265
4.7.1.3	Primitives Exchanged between Data Link Layer and DMPM	266
4.7.1.4	Parameters of DMPM/Data Link Layer Primitives	267
4.7.2	DMPM State Machine	268
4.7.2.1	DMPM States	268
4.7.2.2	DMPM State Table	268
4.7.2.3	Functions used by DMPM	274
4.7.3	Data Link Layer Service Selection	276
4.7.3.1	Introduction	276
4.8	Protocol Options	276
4.8.1	Protocol Option 1	276
4.8.1.1	Introduction	276
4.8.1.2	Abstract Syntax Selection	277
4.8.1.3	Protocol Machine Overview	277
4.8.1.4	Application Relationship Protocol Machine (ARPM) Selection	277
4.8.1.5	Encoding Rule Selection	277
4.8.2	Protocol Option 2	277
4.8.2.1	Introduction	277
4.8.2.2	Abstract Syntax	278
4.8.2.3	Protocol Machine Overview	278
4.8.3	Protocol Option 3	279
4.8.3.1	Introduction	279
4.8.3.2	Abstract Syntax	279
4.8.3.3	Protocol Machine Overview	279
5	Type 2	280
5.1	Type 2 Abstract Syntax	280

5.1.1	FAL PDU Abstract Syntax	280
5.1.1.1	PDU Structure	280
5.1.1.2	UCMM_PDUs	281
5.1.1.3	Transport_Headers	282
5.1.1.4	CM_PDUs	284
5.1.1.5	CM PDU Components	290
5.1.1.6	MR Headers	297
5.1.1.7	OM_Service_PDU	297
5.1.1.8	Message and Connection Paths	306
5.1.1.9	Class, Attribute And Service Codes	312
5.1.1.10	Error Codes	316
5.1.2	Data Abstract Syntax Specification	325
5.1.2.1	Transport Format Specification	325
5.1.2.2	Abstract Syntax Notation	326
5.1.2.3	Control Network Data Specification	326
5.1.2.4	Data Type Specification / Dictionaries	327
5.2	Type 2 Transfer Syntax	330
5.2.1	Compact Encoding	330
5.2.1.1	Encoding Rules	330
5.2.1.2	Encoding Constraints	330
5.2.1.3	Examples (Informative)	330
5.2.2	Data Type Reporting	336
5.2.2.1	Object Data Representation	336
5.2.2.2	Elementary Data Type Reporting	337
5.2.2.3	Constructed Data Type Reporting	337
5.3	Structure of Type 2 FAL Protocol State Machines	341
5.4	Type 2 AP Context State Machine	341
5.5	FAL Service Protocol Machine (FSPM)	341
5.5.1	Primitive Definitions	341
5.5.2	Parameters of Primitives	344
5.5.3	FSPM State Machines	345
5.6	Application Relationship Protocol Machines (ARPMs)	345
5.6.1	Connection-less ARPM (UCMM)	345
5.6.1.1	Primitive Definitions	346
5.6.1.2	Parameters of Primitives	347
5.6.1.3	UCMM State Machines	347
5.6.1.4	Examples of UCMM Sequences (Informative)	353
5.6.1.5	Management UCMM	354
5.6.2	Connection-oriented ARPMs (Transports)	355
5.6.2.1	Transport PDU Buffer	355
5.6.2.2	Transport Classes	355
5.6.2.3	Common Primitive Definitions	356
5.6.2.4	Parameters of Common Primitives	357
5.6.2.5	Transport State Machines – Class 0	357
5.6.2.6	Transport State Machines – Class 1	360
5.6.2.7	Transport State Machines – Class 2	367
5.6.2.8	Transport State Machines – Class 3	374
5.6.2.9	Transport State Machines – Classes 4, 5, 6	382

5.6.2.10	Transport State Machines – Class 4	391
5.6.2.11	Transport State Machines – Class 5	397
5.6.2.12	Transport State Machines – Class 6	410
5.7	DLL Mapping Protocol Machine (DMPM)	429
5.7.1	Introduction	429
5.7.1.1	Link Producer	429
5.7.1.2	Link Consumer	429
5.7.2	Primitive Definitions	429
5.7.2.1	Primitives Exchanged between DMPM and ARPM	429
5.7.2.2	Parameters of ARPM/DMPM Primitives	430
5.7.2.3	Primitives Exchanged between Data Link Layer and DMPM	430
5.7.2.4	Parameters of DMPM/Data Link Layer Primitives	430
5.7.3	DMPM State Machine	431
5.7.3.1	DMPM States	431
5.7.3.2	Functions used by DMPM	432
5.7.4	Data Link Layer Service Selection	432
5.8	Protocol Options	432
6	Type 3	433
6.1	FAL Syntax Description	433
6.1.1	APDU Abstract Syntax	433
6.1.2	Data Types	435
6.2	Transfer Syntax	435
6.2.1	Coding of Basic Data Types	435
6.2.2	Coding Section related to Data Exchange PDUs	436
6.2.2.1	Coding of the Field Outp_Data	436
6.2.2.2	Coding of the Field Inp_Data	436
6.2.3	Coding Section related to Slave Diagnosis PDUs	436
6.2.3.1	Coding of the Field Station_status_1	436
6.2.3.2	Coding of the Field Station_status_2	437
6.2.3.3	Coding of the Field Station_status_3	437
6.2.3.4	Coding of the Field Diag_Master_Add	437
6.2.3.5	Coding of the Field Ident_Number	437
6.2.3.6	Coding of the Field Header_Byte	437
6.2.3.7	Coding of the Field Alarm_Type	438
6.2.3.8	Coding of the Field Status_Type	438
6.2.3.9	Coding of the Field Slot_Number	439
6.2.3.10	Coding of the Field Alarm_Specifier	439
6.2.3.11	Coding of the Field Status_Specifier	439
6.2.3.12	Coding of the Field Diagnosis_User_Data	439
6.2.3.13	Coding of the Field Modul_Status_Array	439
6.2.3.14	Coding of the Field Identifier_Diagnosis_Data_Array	441
6.2.3.15	Coding of the Field Identifier_Number	442
6.2.3.16	Coding of the Field Channel_Number	442
6.2.3.17	Coding of the Field Type_of_Diagnosis	442
6.2.3.18	Coding of the Field Revision_Number	443
6.2.4	Coding Section related to Parameterisation PDU	443
6.2.4.1	Coding of the Field Station_status	443
6.2.4.2	Coding of the Field WD_Fact_1	444

6.2.4.3	Coding of the Field WD_Fact_2.....	444
6.2.4.4	Coding of the Field min_TSDR.....	444
6.2.4.5	Coding of the Field Group_Ident.....	444
6.2.4.6	Coding of the Field User_Prm_Data.....	444
6.2.4.7	Coding of the Field DPV1_Status_1.....	445
6.2.4.8	Coding of the Field DPV1_Status_2.....	445
6.2.4.9	Coding of the Field DPV1_Status_3.....	445
6.2.5	Coding Section related to Configuration PDUs.....	446
6.2.5.1	Coding of the Field Cfg_Identifier.....	446
6.2.5.2	Coding of the Field Special_Cfg_Identifier.....	446
6.2.5.3	Coding of the Fields Length_Byte.....	447
6.2.5.4	Coding of the Field Manufacturer_Specific_Data.....	447
6.2.5.5	Coding of the Field Extended_Length_Byte.....	447
6.2.5.6	Coding of the Field Data_Type.....	448
6.2.6	Coding Section related to Global Control PDUs.....	449
6.2.6.1	Coding of the Field Control_Command.....	449
6.2.6.2	Coding of the Field Group_Select.....	450
6.2.7	Coding Section related to Function Identification and Errors.....	450
6.2.7.1	Coding of the Field Function_Num.....	450
6.2.7.2	Coding of the Field Error_Decompose.....	452
6.2.7.3	Coding of the Field Error_Code_1.....	453
6.2.7.4	Coding of the Field Error_Code_2.....	453
6.2.8	Coding Section related to Master Diagnosis PDU.....	454
6.2.8.1	Coding of the Field MDiag_Identifier.....	454
6.2.8.2	Coding of the Field System_Diagnosis.....	454
6.2.8.3	Coding of the Field USIF_State.....	454
6.2.8.4	Coding of the Field Hardware_Release_DP.....	454
6.2.8.5	Coding of the Field Firmware_Release_DP.....	455
6.2.8.6	Coding of the Field Hardware_Release_User.....	455
6.2.8.7	Coding of the Field Firmware_Release_User.....	455
6.2.8.8	Coding of the Field Data_Transfer_List.....	455
6.2.9	Coding Section related to Upload/Download/Act Para PDUs.....	455
6.2.9.1	Coding of the Field Area_Code_UpDownload.....	455
6.2.9.2	Coding of the Field Timeout.....	456
6.2.9.3	Coding of the Field Max_Len_Data_Unit.....	456
6.2.9.4	Coding of the Field Add_Offset.....	456
6.2.9.5	Coding of the Field Data.....	456
6.2.9.6	Coding of the Field Data_Len.....	456
6.2.9.7	Coding of the Field Area_CodeActBrct.....	456
6.2.9.8	Coding of the Field Area_CodeAct.....	456
6.2.9.9	Coding of the Field Activate.....	457
6.2.10	Coding Section related to the Bus Parameter Set.....	457
6.2.10.1	Coding of the Field Bus_Para_Len.....	457
6.2.10.2	Coding of the Field DL_Add.....	457
6.2.10.3	Coding of the Field Baud_rate.....	457
6.2.10.4	Coding of the Fields T _{SL} , min T _{SDR} , max T _{SDR}	458
6.2.10.5	Coding of the Fields T _{QUI} , T _{SET} , G, HSA, max_retry_limit.....	458

6.2.10.6	Coding of the Field T_{TR} (Target Token Rotation Time).....	458
6.2.10.7	Coding of the Field Bp_Flag (Busparameter Flag)	458
6.2.10.8	Coding of the Field Min_Slave_Interval.....	458
6.2.10.9	Coding of the Field Poll_Timeout.....	458
6.2.10.10	Coding of the Field Data_Control_Time	458
6.2.10.11	Coding of the Field Alarm_Max	458
6.2.10.12	Coding of the Field Max_User_Global_Control	458
6.2.10.13	Coding of the Field Master_User_Data_Len	459
6.2.10.14	Coding of the Field Master_Class2_Name	459
6.2.10.15	Coding of the Field Master_User_Data	459
6.2.11	Coding Section related to the Slave Parameter Set.....	459
6.2.11.1	Coding of the Field Slave_Para_Len.....	459
6.2.11.2	Coding of the Field SI_Flag (Slave Flag).....	459
6.2.11.3	Coding of the Field Slave_Type.....	460
6.2.11.4	Coding of the Field Max_Diag_Data_Len.....	460
6.2.11.5	Coding of the Field Max_Alarm_Len.....	460
6.2.11.6	Coding of the Field Max_Channel_Data_Length.....	460
6.2.11.7	Coding of the Field Diag_Upd_Delay	460
6.2.11.8	Coding of the Field Alarm_Mode	460
6.2.11.9	Coding of the Field Add_SI_Flag.....	460
6.2.11.10	Coding of the Field Prm_Data_Len.....	461
6.2.11.11	Coding of the Field Prm_Data	461
6.2.11.12	Coding of the Field Cfg_Data_Len.....	461
6.2.11.13	Coding of the Field Cfg_Data.....	461
6.2.11.14	Coding of the Field Add_Tab_Len.....	461
6.2.11.15	Coding of the Field Number_of_Entries	461
6.2.11.16	Coding of the Field Add_Tab_Entry_Header	461
6.2.11.17	Coding of the Field I/O_Data_Length.....	461
6.2.11.18	Coding of the Field I/O_Config_Address.....	461
6.2.11.19	Coding of the Field Host_Address	461
6.2.11.20	Coding of the Field Slave_User_Data_Len	462
6.2.11.21	Coding of the Field Slave_User_Data	462
6.2.12	Coding Section related to Statistic Counters	462
6.2.12.1	Coding of the Field Frame_sent_count and SD_count.....	462
6.2.12.2	Coding of the Field Error_count and SD_error_count.....	462
6.2.13	Coding Section related to Set Slave Address PDU.....	462
6.2.13.1	Coding of the Field New_Slave_Add.....	462
6.2.13.2	Coding of the Field No_Add_Change.....	462
6.2.13.3	Coding of the Field Rem_Slave_Data	462
6.2.14	Coding Section related to Initiate/Abort PDUs.....	462
6.2.14.1	Coding of the Field Features_Supported_1.....	462
6.2.14.2	Coding of the Field Features_Supported_2.....	462
6.2.14.3	Coding of the Field Profile_Features_Supported_1	462
6.2.14.4	Coding of the Field Profile_Features_Supported_2	463
6.2.14.5	Coding of the Field Profile_Ident_Number	463
6.2.14.6	Coding of the Field S_Type (Source Type)	463
6.2.14.7	Coding of the Field D_Type (Destination Type).....	463
6.2.14.8	Coding of the Field S_Len (Source Length)	463

6.2.14.9	Coding of the Field D_Len (Destination Length).....	463
6.2.14.10	Coding of the Field S_API (Source Application Identifier).....	463
6.2.14.11	Coding of the Field D_API (Destination Application Identifier).....	463
6.2.14.12	Coding of the Field S_SCL (Source Security Level).....	463
6.2.14.13	Coding of the Field D_SCL (Destination Security Level).....	463
6.2.14.14	Coding of the Field S_Network_Address.....	463
6.2.14.15	Coding of the Field D_Network_Address.....	463
6.2.14.16	Coding of the Field S_MAC_Address.....	464
6.2.14.17	Coding of the Field D_MAC_Address.....	464
6.2.14.18	Coding of the Field Send_Timeout.....	464
6.2.14.19	Coding of the Field Server_SAP.....	464
6.2.14.20	Coding of the Field Subnet.....	464
6.2.14.21	Coding of the Field Instance_Reason_Code.....	464
6.2.15	Coding Section related to Read/Write/Data Transport PDUs.....	465
6.2.15.1	Coding of the Field Index.....	465
6.2.15.2	Coding of the Field Length.....	465
6.2.16	Example of Diagnosis-RES-PDU.....	466
6.2.17	Example of Chk_Cfg-REQ-PDU.....	467
6.2.18	Example of Chk_Cfg-REQ-PDU with Extension.....	467
6.2.19	Example Structure of the Data_Unit for Data_Exchange.....	468
6.3	FAL Protocol State Machines.....	469
6.3.1	Overall Structure.....	469
6.3.1.1	Fieldbus Service Protocol Machines (FSPM).....	469
6.3.1.2	Master to Slave Cyclic (MS0).....	469
6.3.1.3	Master (Class 1) to Slave Acyclic (MS1).....	469
6.3.1.4	Master (Class 2) to Slave Acyclic (MS2).....	469
6.3.1.5	Master Master Acyclic (MM1/MM2).....	469
6.3.1.6	DLL Mapping Protocol Machines (DMPM).....	469
6.3.2	Assignment of State Machines to Devices.....	470
6.3.3	Overview DP-Slave.....	471
6.3.4	Overview DP-Master (Class 1).....	472
6.3.5	Overview DP-Master (Class 2).....	473
6.3.6	Cyclic Communication between DP-Master (Class 1) and DP-Slave.....	473
6.3.7	Acyclic Communication between DP-Master (Class 2) and DP-Master (Class 1).....	475
6.3.8	Application Relationship Monitoring.....	477
6.3.8.1	Monitoring of the MS0 - AR.....	477
6.3.8.2	Monitoring of the MS2 - AR.....	478
6.4	AP Context State Machine.....	480
6.5	FAL Service Protocol Machines (FSPMs).....	480
6.5.1	FSPMS.....	480
6.5.1.1	Primitive Definitions.....	480
6.5.1.2	State Machine Description.....	483
6.5.1.3	FSPMS State Table.....	486
6.5.1.4	Functions.....	495
6.5.2	FSPMM1.....	496
6.5.2.1	Primitive Definitions.....	496
6.5.2.2	State Machine Description.....	499
6.5.2.3	FSPMM1 State Table.....	501

6.5.2.4	Functions	514
6.5.3	FSPMM2	514
6.5.3.1	Primitive Definitions	514
6.5.3.2	State Machine Description	516
6.5.3.3	FSPMM2 State Table	517
6.5.3.4	Functions	521
6.6	Application Relationship Protocol Machines (ARPMs)	521
6.6.1	MSCY1S	521
6.6.1.1	Primitive Definitions	521
6.6.1.2	State Machine Description	522
6.6.1.3	MSCY1S State Table	526
6.6.1.4	Functions	539
6.6.2	MSAC1S	541
6.6.2.1	Primitive Definitions	541
6.6.2.2	State Machine Description	542
6.6.2.3	MSAC1S State Table	544
6.6.2.4	Functions	551
6.6.3	MSRM2S	551
6.6.3.1	Primitive Definitions	551
6.6.3.2	State Machine Description	551
6.6.3.3	MSRM2S State Table	554
6.6.4	MSAC2S	556
6.6.4.1	Primitive Definitions	556
6.6.4.2	State Machine Description	557
6.6.4.3	MSAC2S State Table	560
6.6.5	MSCY1M	569
6.6.5.1	Primitive Definitions	569
6.6.5.2	State Machine Description	570
6.6.5.3	MSCY1M State Table	573
6.6.6	MSAL1M	583
6.6.6.1	Primitive Definitions	583
6.6.6.2	State Machine Description	584
6.6.6.3	MSAL1M State Table	587
6.6.7	MSAC1M	591
6.6.7.1	Primitive Definitions	591
6.6.7.2	State Machine Description	592
6.6.7.3	MSAC1M State Table	597
6.6.8	MMAC1	601
6.6.8.1	Primitive Definitions	601
6.6.8.2	State Machine Description	601
6.6.8.3	MMAC1 State Table	602
6.6.9	MSAC2M	606
6.6.9.1	Primitive Definitions	606
6.6.9.2	State Machine Description	608
6.6.9.3	MSAC2M State Table	611
6.6.10	MMAC2	619
6.6.10.1	Primitive Definitions	619
6.6.10.2	State Machine Description	620

6.6.10.3	MMAC2 State Table	621
6.7	DLL Mapping Protocol Machines (DMPMs)	625
6.7.1	DMPMS	625
6.7.1.1	Primitive Definitions	625
6.7.1.2	State Machine Description	628
6.7.1.3	DMPMS State Table	629
6.7.1.4	Functions	632
6.7.2	DMPMM1	633
6.7.2.1	Primitive Definitions	633
6.7.2.2	State Machine Description	637
6.7.2.3	DMPMM1 State Table	638
6.7.2.4	Functions	642
6.7.3	DMPMM2	642
6.7.3.1	Primitive Definitions	642
6.7.3.2	State Machine Description	645
6.7.3.3	DMPMM2 State Table	646
6.7.3.4	Functions	648
6.8	Protocol Options	649
6.9	Parameters for a DP-Slave	649
7	Type 4	650
7.1	FAL Syntax Description	650
7.1.1	FAL-AR PDU Abstract Syntax	650
7.1.1.1	Abstract Syntax of APDU Header	650
7.1.1.2	Abstract Syntax of APDU Body	650
7.1.2	Data Types	651
7.2	Transfer Syntaxes	652
7.2.1	APDU Encoding	652
7.2.1.1	APDU Header Encoding	652
7.2.1.2	APDU Body Encoding	654
7.2.2	Variable Object Encoding and Packing	656
7.2.2.1	Encoding of Simple Variables	656
7.2.2.2	Encoding of Constructed Variables	656
7.2.2.3	Alignment	656
7.2.2.4	Variable Object Attributes	657
7.2.3	Error Codes	658
7.3	FAL Protocol State Machines	659
7.4	AP Context State Machine	660
7.5	FAL Service Protocol Machine (FSPM)	661
7.5.1	Primitives Exchanged between FAL User and FSPM	661
7.5.2	FSPM States	661
7.5.2.1	FSPM Proxy Object States	661
7.5.2.2	FSPM Real Object State Machine Description	665
7.6	Application Relationship Protocol Machine (ARPM)	666
7.6.1	Primitives Exchanged between ARPM and FSPM	666
7.6.2	ARPM States	667
7.6.2.1	Overview	667
7.6.2.2	Sender State Transitions	667
7.6.2.3	Receiver State Transitions	668

7.7	DLL Mapping Protocol Machine (DMPM)	669
7.7.1	Data Link Layer Service Selection	669
7.7.1.1	Introduction	669
7.7.2	Primitives Exchanged between ARPM and DLPM	669
7.7.3	Primitives Exchanged between DLPM and Data Link Layer	669
7.7.4	DLPM States	670
7.7.4.1	Overview	670
7.7.4.2	Sender State Transitions	670
7.7.4.3	Receiver State Transitions	671
7.8	Protocol Options	672
8	Type 5	673
8.1	FAL Syntax Description	673
8.1.1	PDU Abstract Syntax	673
8.1.2	Abstract Syntax of Data Types	673
8.2	Transfer Syntax	673
8.2.1	Data Type Encodings	673
8.2.2	Type 5 Header	674
8.2.2.1	Conveyed Reference Use	675
8.2.3	Type 5 Trailer	679
8.2.4	APDU Body	679
8.2.4.1	Type 5 AR ASE Services	680
8.2.4.2	SMK ASE Services	680
8.2.4.3	Type 1 ASE Services	684
8.2.4.4	LAN Redundancy Services	698
8.2.4.5	List of Type 1 Managed AP Attributes	698
8.2.5	Error APDU Bodies	703
8.2.5.1	Common Error Parameters	703
8.2.5.2	FI Error Parameters	703
8.2.5.3	OD Error Parameters	704
8.3	FAL Protocol State Machine Structure	704
8.4	AP Context State Machine	704
8.5	FAL Service Protocol Machine (FSPM)	704
8.6	Application Relationship Protocol Machines (ARPMs)	704
8.6.1	Client / Server Session ARPM	704
8.6.1.1	Primitive Definitions	704
8.6.1.2	DLL Mapping of Client / Server AREP Class	705
8.6.1.3	Client / Server AREP State Machine	706
8.6.2	Publisher / Subscriber Session ARPM	717
8.6.2.1	Primitive Definitions	717
8.6.2.2	DLL Mapping of Publisher / Subscriber AREP Class	717
8.6.2.3	Publisher / Subscriber ARPM State Machine	718
8.7	DLL Mapping Protocol Machine (DMPM)	723
8.7.1	Primitive Definitions	723
8.7.1.1	Primitives Exchanged between DMPM and ARPM	723
8.7.1.2	Parameters of ARPM/DMPM Primitives	723
8.7.1.3	Primitives Exchanged between the Socket model and DMPM	723
8.7.1.4	Parameters of DMPM/Socket model Primitives	724
8.7.2	DMPM State Machine	724

8.7.2.1	DMPM States.....	724
8.7.2.2	DMPM State Table.....	724
8.7.2.3	Functions used by DMPM.....	725
8.8	Protocol Options	726
9	Type 6.....	727
10	Type 7.....	728
10.1	Abstract syntax of data type for Type 7	728
10.1.1	Boolean	728
10.1.2	Integer.....	728
10.1.3	Bit string	728
10.1.4	Octet string.....	729
10.1.5	Sequence (sequence or sequence of)	729
10.1.6	Choice	730
10.1.7	Null.....	730
10.1.8	Object identifier	730
10.1.9	Default.....	731
10.1.10	Example of encoding (informative)	731
10.2	Transfer Syntaxes.....	732
10.2.1	MPS ASE FAL PDU data types.....	733
10.2.1.1	Syntax notation	733
10.2.1.2	Simple predefined types	734
10.2.1.3	PDU Types	734
10.2.2	MCS AR ASE AL PDU types.....	744
10.2.2.1	Coding rules	744
10.2.2.2	Structure of the PDUs.....	745
10.2.3	SUB MMS ASE FAL PDU type.....	756
10.2.3.1	SUB-MMS-PDU	756
10.2.4	VMD ASE Protocol.....	767
10.2.4.1	Environment management protocol	767
10.2.4.2	Protocol concerning the vmd management	771
10.2.5	Domain ASE protocol.....	772
10.2.5.1	Introduction.....	772
10.2.5.2	Definition of the confirmed delete domain service protocol.....	773
10.2.5.3	Definition of the confirmed initiate download sequence service protocol.....	773
10.2.5.4	Definition of the confirmed upload segment service protocol.....	775
10.2.5.5	Definition of the confirmed terminate upload sequence service protocol.....	775
10.2.5.6	Definition of the confirmed get domain attributes service protocol.....	775
10.2.6	Program Invocation ASE protocol.....	776
10.2.6.1	Introduction.....	776
10.2.6.2	Definition of the confirmed create program invocation service protocol	776
10.2.6.3	Definition of the confirmed delete program invocation service protocol.....	777
10.2.6.4	Definition of the confirmed start service protocol	778
10.2.6.5	Definition of the confirmed stop service protocol.....	778
10.2.6.6	Definition of the confirmed resume service protocol.....	778
10.2.6.7	Definition of the confirmed reset service protocol	778
10.2.6.8	Definition of the confirmed kill service protocol.....	778
10.2.6.9	definition of the confirmed service protocol get program invocation attributes	778

10.2.7	Variable ASE protocol	779
10.2.7.1	Common definitions	779
10.2.7.2	Definition of the confirmed read service protocol	780
10.2.7.3	Definition of the confirmed write service protocol	781
10.2.7.4	Definition of the unconfirmed information report service protocol	782
10.2.7.5	Definition of the confirmed define variable list service protocol	782
10.2.7.6	Definition of the confirmed delete variable list service protocol	783
10.2.7.7	Definition of the confirmed get variable access attributes service protocol	784
10.2.7.8	Definition of the confirmed get variable list attributes service protocol	784
10.2.8	Event ASE protocol	785
10.2.8.1	Introduction	785
10.2.8.2	Definition of the unconfirmed event notification service protocol	786
10.2.8.3	Definition of the confirmed acknowledge event notification service protocol	787
10.2.8.4	Definition of the confirmed alter event condition monitoring service protocol	787
10.2.8.5	Definition of the confirmed get alarm summary service protocol	788
10.2.8.6	Definition of the confirmed get event condition attribute service protocol	789
10.2.9	Directory ASE Protocol	790
10.3	Structure of Protocol Machines	790
10.4	AP-Context state machine	792
10.5	Sub-MMS FAL Service Protocol Machine (FSPM)	792
10.5.1	Projection of the SUB-MMS PDUs on the MCS services	792
10.5.2	Projection of the SUB-MMS abort service on the MCS services	792
10.5.3	Construction of a SUB-MMS-PDU from a service primitive	792
10.5.4	Extraction of a valid service primitive from a SUB-MMS-PDU	793
10.5.5	Negotiation of an abstract syntax and a transfer syntax commonly called presentation-context	793
10.5.6	Identification of the SUB-MMS core abstract syntax	795
10.5.7	Identification of the application context name	795
10.5.8	Identification of the ASE of the core abstract syntax and the transfer syntax	795
10.6	DLL Mapping Protocol Machine (DMPM) and Association Relationship Protocol Machine (ARPM)	796
10.6.1	MPS ARPM and DMPM	796
10.6.1.1	Service procedures	796
10.6.1.2	Service elements procedures	803
10.6.2	MCS ARPM and DMPM	805
10.6.2.1	Association establishment	806
10.6.2.2	Association termination	809
10.6.2.3	Association revocation	814
10.6.2.4	Associated mode data transfer	816
10.6.2.5	Non-associated mode data transfer	837
10.7	Protocol options	841
10.7.1	Conformances classes	841
10.7.1.1	Vertical conformance	842
10.7.1.2	Horizontal conformance	843
10.7.1.3	Type conformance	847
10.7.1.4	Protocol Implementation Conformance statement	849
11	Type 8	859
11.1	FAL Syntax Description	859

11.1.1	FAL-AR PDU Abstract Syntax.....	859
11.1.1.1	Confirmed Send Service	859
11.1.1.2	Unconfirmed Send Service	859
11.1.1.3	Unconfirmed Acknowledge Send Service.....	859
11.1.1.4	Establish Service	859
11.1.1.5	ConType.....	860
11.1.2	Abstract Syntax of PDUBody	860
11.1.2.1	Abort Service	860
11.1.2.2	ConfirmedServiceRequest.....	860
11.1.2.3	ConfirmedServiceResponse	860
11.1.2.4	ConfirmedServiceError.....	861
11.1.2.5	Error Type	861
11.1.2.6	Error Fi Type	861
11.1.2.7	Error Class	861
11.1.2.8	Unconfirmed PDUs	861
11.1.2.9	Management ASE	861
11.1.2.10	Application Process ASE.....	861
11.1.2.11	Function Invocation ASE	862
11.1.2.12	Variable ASE	863
11.1.3	Type definitions for ASEs.....	863
11.1.3.1	AP ASE Types	863
11.1.3.2	AR ASE Types	863
11.1.3.3	Function Invocation ASE Types.....	863
11.1.3.4	Management ASE Types.....	864
11.1.3.5	Variable ASE Types.....	864
11.1.3.6	General Types	864
11.1.3.7	Object Definitions	865
11.1.4	Abstract Syntax of Data Types	866
11.1.4.1	Notation for BinaryDate2000 Type.....	867
11.2	Transfer Syntax	867
11.2.1	Peripherals Encoding Rules (PER).....	867
11.2.2	Encoding of APDU types	867
11.2.2.1	Overview of APDU encoding	867
11.2.2.2	APDU Header encoding	867
11.2.2.3	APDU Body encoding.....	869
11.2.3	Encoding of tagged type values.....	869
11.2.3.1	Encoding of tagged type values of tag class PRIVATE.....	869
11.2.3.2	Encoding of context specific tagged type values	870
11.2.4	Encoding of simple values.....	871
11.2.4.1	Encoding of a Boolean value	871
11.2.4.2	Encoding of String values (visible string, octet string, bit string).....	871
11.2.4.3	Encoding of an Integer value.....	871
11.2.4.4	Encoding of an Unsigned value	872
11.2.4.5	Encoding of a floating-point value.....	872
11.2.4.6	Encoding of a BinaryDate value.....	872
11.2.4.7	Encoding of a BinaryDate2000 value	872
11.2.4.8	Encoding of a Time of Day value.....	873
11.2.4.9	Encoding of a Time Difference value	873

11.2.4.10	Encoding of a Time value	874
11.2.4.11	Encoding of a Null value	874
11.2.4.12	Encoding of an ANY value	874
11.2.4.13	Encoding of Structured Types	875
11.2.4.14	Encoding of a SEQUENCE value	875
11.2.4.15	Encoding of a SEQUENCE OF value	875
11.2.4.16	Encoding of a CHOICE value	875
11.2.4.17	Encoding of an ENUMERATED value	875
11.2.4.18	Encoding of an Object-Definition value	875
11.3	FAL Protocol State Machine Structure	875
11.4	AP Context State Machine	875
11.5	FAL Service Protocol Machine (FSPM)	875
11.6	Application Relationship Protocol Machines (ARPMs)	876
11.6.1	Queued Usertriggered Bidirectional-Flow Control (QUB-FC) ARPM	876
11.6.1.1	QUB-FC Primitive Definitions	876
11.6.1.2	DLL Mapping of QUB-FC AREP Class	876
11.6.1.3	Attributes	877
11.6.1.4	DLL services	878
11.6.1.5	QUB-FC ARPM State Machine	878
11.6.2	Buffered Networkscheduled Unidirectional (BNU) ARPM	878
11.6.2.1	Primitive definitions	878
11.6.2.2	DLL Mapping of BNU AREP Class	878
11.6.2.3	Attributes	879
11.6.2.4	BNU AREP state machine	879
11.7	DLL Mapping Protocol Machine	879
11.7.1	Overview	879
11.7.2	Primitive definitions	880
11.7.2.1	Primitives Exchanged between DMPM and ARPM	880
11.7.2.2	Parameters of ARPM/DMPM primitives	881
11.7.2.3	Primitives Exchanged between Data Link Layer and DMPM	881
11.7.2.4	Parameters of DMPM/Data Link Layer Primitives	882
11.7.3	DMPM State Machine	882
11.7.3.1	DMPM States	882
11.7.3.2	DMPM State Table	882
11.7.3.3	Functions used by Type-8 DMPM	888
11.8	Protocol Option	890
11.8.1	Introduction	890
11.8.2	Abstract Syntax	890
11.8.3	Protocol machine overview	890
11.8.3.1	Application Relationship Protocol Machine (ARPM) selection	890
11.8.3.2	Encoding Rule Selection	890
11.8.3.3	DLL Mapper selection	890

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**DIGITAL DATA COMMUNICATIONS FOR MEASUREMENT AND CONTROL –
FIELDBUS FOR USE IN INDUSTRIAL CONTROL SYSTEMS –****Part 6: Application Layer protocol specification**

FOREWORD

- 1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the 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, the IEC publishes International Standards. 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. The 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 the 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 National Committees.
- 3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
- 4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
- 5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.
- 6) Attention is drawn to the possibility that some of the elements of this standard may be the subject of patent rights. The IEC shall not be held responsible for identifying any or all such patent rights.

International standard IEC 61158-6 has been prepared by subcommittee 65C: Digital communications, of IEC technical committee 65: Industrial-process measurement and control.

This second edition cancels and replaced the first edition which was issued as a technical specification in 1999. It constitutes a technical revision and now has the status of an International Standard.

This second edition adds seven distinct sets of services, each with a corresponding protocol, to the set of services and protocols of the first edition.

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

FDIS	Report on voting
65C/226/FDIS	65C/231/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 the ISO/IEC Directives, Part 3.

IEC 61158 consists of the following parts, under the general title *Digital data communications for measurement and control — Fieldbus for use in industrial control systems*:

- Part 1: Introductory guide (under preparation)
- Part 2: Physical layer specification and service definition
- Part 3: Data Link Service definition
- Part 4: Data Link Protocol specification
- Part 5: Application Layer service definition
- Part 6: Application Layer protocol specification
- Part 7: System management (under consideration)
- Part 8: Conformance testing (under consideration)

The committee has decided the the contents of this publication will remain unchanged until 2006. At this date, the publication will be

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

INTRODUCTION

This standard describes the Fieldbus Application Layer (FAL) protocol that defines the information interchange and the interactions between Application Entity invocations (AE-Is) to support the services defined in IEC 61158-5.

An Application Process uses the Fieldbus Application Layer services to exchange information with other Application Processes. The services define the abstract interface between the application process and the Application Layer.

The Application Layer protocol is the set of rules that governs the format and meaning of the information exchange between the Application Layers in various devices. The Application Layer uses the protocol to implement the Application Layer services definitions.

The protocol machine defines the various states of an Application Layer and the valid transitions between the states. It may be considered as a finite state machine. The protocol machine is described using state tables. The information is exchanged between the application process and the protocol machine through application service data units. The protocol machine exchanges information with other protocol machines through application protocol data units (APDU).

This set of Application Layer standards does not specify individual implementations or products, nor does it constrain the implementations of Application Entities (AEs) and interfaces within the industrial automation system.

This set of Application Layer standards does not contain test procedures to ensure compliance with such requirements.

DIGITAL DATA COMMUNICATIONS FOR MEASUREMENT AND CONTROL – FIELDBUS FOR USE IN INDUSTRIAL CONTROL SYSTEMS –

Part 6: Application Layer protocol specification

1 Scope

The Fieldbus Application Layer (FAL) is an Application Layer communication standard designed to support the conveyance of time-critical application requests and responses among devices in an automation environment. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard specifies interactions between remote applications in terms of

- the encoding rules that are applied to all the Application Layer Protocol Data Units (APDUs);
- the formal Abstract Syntax definitions of such APDUs;
- the protocol state machine descriptions that handle the APDUs and the primitives in the correct sequences;
- the mappings of the APDUs to and from the Data Link Layer services defined in IEC 61158-3.

The FAL encoding rules are designed assuming that both the encoder (sender) and the decoder (receiver) have the common knowledge of the abstract syntax. Wherever possible, data types identifiers are not encoded and transferred over the network.

NOTE This is why the Abstract Syntax Notation One / Basic Encoding Rule is not practical for the FAL.

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

- a) to the Fieldbus Data Link Layer at the boundary between the Application and Data Link Layers of the Fieldbus Reference Model, and
- b) to the System Management at the boundary between the System Management and Application Layers of the Fieldbus Reference Model.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of IEC 61158. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of IEC 61158 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of IEC and ISO maintain registers of currently valid International Standards

IEC 61131-3:1993, *Programmable controllers – Part 3: Programming languages*

IEC 61158-2:1993, *Fieldbus Standard for use in industrial control systems – Part 2: Physical layer specification and service definition*

IEC 61158-3:2000, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 3: Data Link service definitions*

IEC 61158-4:2000, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 4: Data Link protocol specifications*

IEC 61158-5:2000, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Part 5: Application Layer service definitions*

ISO/IEC 7498:all parts, *Information technology – Open Systems Interconnection – Basic Reference Model*

ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 8822:1994, *Information technology – Open Systems Interconnection – Presentation service definition*

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

ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

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

ISO/IEC 9506-2:1990, *Industrial automation systems – Manufacturing message specification – Part 2: Protocol specification*

ISO/IEC 9545:1994, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 10646: all parts, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*

ISO/IEC 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane*

ISO/IEC 10731:1994, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ISO 8649:1996, *Information technology – Open Systems Interconnection – Service definition for the Association Control Service Element Protocol specification*

ISO 8650: all parts, *Information technology – Open Systems Interconnection – Connection-oriented protocol for the Association Control Service Element Protocol specification*

IEEE 754:1985, *Binary Floating-Point Arithmetic*

3 Definitions

For the purpose of this standard, the following definitions apply:

3.1 Definitions from other ISO/IEC Standards

3.1.1 Definitions from ISO/IEC 7498-1

- 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 Definitions from ISO/IEC 8822

- a) abstract syntax;
- b) presentation context.

3.1.3 Definitions from ISO/IEC 9545

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

3.1.4 Definitions from ISO/IEC 8824

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

3.1.5 Definitions from ISO/IEC 8825

- a) encoding (of a data value);
- b) data value;
- c) Identifier Octets (the singular form is used in this standard);
- d) Length Octet(s) (both singular and plural forms are used in this standard);
- e) Contents Octets.

3.2 Definitions from IEC 61158-5

- a) application relationship;
- b) conveyance path;
- c) client;
- d) dedicated AR;

- e) dynamic AR;
- f) error class;
- g) error code;
- h) name;
- i) numeric identifier;
- j) peer;
- k) pre-defined AR endpoint;
- l) pre-established AR endpoint;
- m) publisher;
- n) server.

3.3 Other definitions

NOTE Type 1 definitions shall apply to all the types except if the same definition is used with other types, in that case the latter shall have precedence.

The following definitions are used in this standard:

3.3.1 Type 1 Definitions

3.3.1.1 called

Service user or a service provider that receives an indication primitive or a request APDU.

3.3.1.2 calling

Service user or a service provider that initiates a request primitive or a request APDU.

3.3.1.3 interoperability

Capability of User Layer entities to perform coordinated and cooperative operations using the services of the FAL.

3.3.1.4 management information

Network accessible information that supports the management of the Fieldbus environment.

3.3.1.5 receiving

Service user that receives a confirmed primitive or an unconfirmed primitive, or a service provider that receives a confirmed APDU or an unconfirmed APDU.

3.3.1.6 resource

Resource is a processing or information capability of a subsystem.

3.3.1.7 sending

Service user that sends a confirmed primitive or an unconfirmed primitive, or a service provider that sends a confirmed APDU or an unconfirmed APDU.

3.3.2 Type 2 Definitions

3.3.2.1 Allocate

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

3.3.2.2 Application

Function or data structure for which data is consumed or produced.

3.3.2.3 Application objects

Multiple object classes that manage and provide the run time exchange of messages across the network and within the network device.

3.3.2.4 Attribute

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

3.3.2.5 Behaviour

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

3.3.2.6 Class

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

3.3.2.7 Class attributes

An attribute that is shared by all objects within the same class.

3.3.2.8 Class code

A unique identifier assigned to each object class.

3.3.2.9 Class specific service

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

3.3.2.10 Client

- (1) An object which uses the services of another (server) object to perform a task.
- (2) An initiator of a message to which a server reacts.

communication objects Components that manage and provide run time exchange of messages across the network such as the Connection Manager object, the unconnected message manager (UCMM), and the Message Router object.

3.3.2.11 Connection

A logical binding between two application objects. These application objects may be within the same or different devices.

3.3.2.12 Connection ID (CID)

Identifier assigned to a transmission that is associated with a particular connection between producers and consumers that identifies a specific piece of application information.

3.3.2.13 Connection path

The attribute is made up of a byte stream which defines the application object to which a connection instance applies.

3.3.2.14 Connection point

A buffer that is part of another object. The buffer is represented as a subinstance of an assembly object.

3.3.2.15 Consume

The act of receiving data from a producer.

3.3.2.16 Consumer

A node that is receiving data from a producer.

3.3.2.17 Consuming application

The application that consumes data.

3.3.2.18 Cyclic

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

3.3.2.19 Device

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

3.3.2.20 Device profile

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

3.3.2.21 End node

A producing or consuming node.

3.3.2.22 End point

One of the communicating entities involved in a connection.

3.3.2.23 Error

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

3.3.2.24 Frame

Single data transfer on a link.

3.3.2.25 Instance

The actual physical occurrence of an object within a class. Identifies one of many objects within the same object class. For example: California is an instance of the object class state. The terms object, instance, and object instance are used to refer to a specific instance.

3.3.2.26 Instance attributes

An attribute that is unique to an object instance and not shared by the object class.

3.3.2.27 Instantiated

An object that has been created in a device.

3.3.2.28 Keeper

Object responsible for distributing link configuration data to all nodes on the link.

3.3.2.29 Little endian

Describes a model of memory organisation which stores the least significant byte at the lowest address. On the network medium, the lowest order byte is transferred first.

3.3.2.30 Lpacket

The Lpacket (or link packet) is a piece of application information that contains a size, control byte, tag, and link data. Peer Data Link Layers use Lpackets to send and receive service data units from higher layers in the OSI stack.

3.3.2.31 Member

A piece of an attribute that is structured as an array.

3.3.2.32 Message router

The object within a node that distributes messaging requests to the appropriate application objects.

3.3.2.33 Multicast connection

A connection from one node to many. Multicast connections allow messages from a single producer to be received by many consumer nodes.

3.3.2.34 Network

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

3.3.2.35 Object

An abstract representation of a particular component within a device, i.e. :

(1) An abstract representation of a computer's capabilities. Objects can be composed of any or all of the following components:

- a) data (information which changes with time);

- b) configuration (parameters for behaviour);
- c) methods (things that can be done using data and configuration).

(2) A collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behaviour.

3.3.2.36 Object specific service

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

3.3.2.37 Originator

The client responsible for establishing a connection path to the target.

3.3.2.38 Point-to-point connection

A connection that exists between two nodes only. Connections can be either point-to-point or multicast.

3.3.2.39 Produce

Act of sending data to be received by a consumer.

3.3.2.40 Producer

A node that is responsible for sending data.

3.3.2.41 Server

An object which provides services to another (client) object.

3.3.2.42 Service

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

3.3.2.43 Target

The end-node to which a connection is established.

3.3.2.44 Temporary node

Same as transient node.

3.3.2.45 Transaction id

Field within the UCMM header that matches a response with the associated request. The server echoes this field in the response message.

3.3.2.46 Unconnected message manager (UCMM)

The component within a node that transmits and receives unconnected explicit messages and sends them directly to the Message Router object.

3.3.2.47 Unconnected service

The messaging service which does not rely on the set up of a connection between devices before allowing information exchanges.

3.3.3 Type 3 Definitions

3.3.3.1 Access Protection

The access to certain functions of a DP-Slave is restricted to one DP-Master.

3.3.3.2 Address-assignment-table

Mapping of the DP-Masters internal IO-addresses to the decentralised inputs and outputs.

3.3.3.3 Channel

A single physical or logical connection of inputs or outputs of a DP-Slave to the process.

3.3.3.4 Channel related Diagnosis

This diagnosis information refers to a single input or output channel of the DP-Slave.

3.3.3.5 Configuration check

In the start-up phase, the DP-Master transmits the expected configuration for comparison with the real configuration to the DP-Slave.

3.3.3.6 Configuration fault

If the DP-Slave detects a not acceptable difference between the expected configuration and the real configuration, then a configuration fault is indicated in the diagnosis information.

3.3.3.7 Configuration Identifier

For each input- and/or output-module of a DP-Slave at least one Configuration Identifier is defined in the configuration string.

3.3.3.8 Configuration Identifier related Diagnosis

This diagnosis information is related to the input- or output-range for which a Configuration Identifier is defined in the configuration string.

3.3.3.9 Control Commands

Control commands will be transferred from the DP-Master to the DP-Slave. With the control commands it is possible to clear the outputs, or to Freeze the inputs and/or synchronise the outputs of the DP-Slaves.

3.3.3.10 Data consistency

The input- or output-data-range that always shall be transmitted consistent between the DP-Master and the DP-Slave.

3.3.3.11 Default address

The default address of a DP-Slave is 126.

This address will be used as initial value if address setting will be done via PROFIBUS-DP.

3.3.3.12 Device related diagnosis

This diagnosis information is related to the whole Slave.

3.3.3.13 Diagnosis information

This term is used to identify the whole diagnosis data of a DP-Slave.

3.3.3.14 Diagnosis information collection

A system-diagnosis information that is collected at the DP-Master (Class 1).

3.3.3.15 DP-Master (Class 1)

A device that controls the assigned DP-Slaves; usually a programmable controller or distributed control system.

3.3.3.16 DP-Master (Class 2)

A device that interacts as a configuration or diagnosis tool; usually a programming device.

3.3.3.17 DP-Slave

A device that interacts to one DP-Master as a provider for cyclic I/O data exchange. In addition acyclic functions could be provided.

3.3.3.18 Freeze

By means of this function the input data of several DP-Slaves will be simultaneously frozen.

3.3.3.19 Group

A set of DP-Slaves which perform a Freeze or Sync operation.

3.3.3.20 I/O data

Data which have to be transferred cyclically for the purpose of processing

3.3.3.21 Ident Number

The Ident Number specifies a DP-Master (Class 1) or DP-Slave device type.

3.3.3.22 Index

Address of an object within an application process.

3.3.3.23 Master parameter set

The Master parameter set includes the configuration and parameterisation data of all DP-Slaves that are assigned to the corresponding DP-Master and the bus parameters.

3.3.3.24 Module

Addressable unit inside the DP-Slave.

3.3.3.25 Process data

Data which are already pre-processed and transferred acyclically for the purpose of information or further processing.

3.3.3.26 Real Configuration

The real configuration describes the input- and output-data-ranges of the DP-Slave. Additionally, the definition of the data consistency is included in the real configuration.

3.3.3.27 Slot

Address of a module within a DP-Slave.

3.3.3.28 Sync

By means of this function the outputs of several DP-Slaves will be simultaneously updated.

3.3.4 Type 4 Definitions

There are no Type 4 definitions.

3.3.5 Type 5 Definitions

There are no Type 5 definitions.

3.3.6 Type 6 Definitions

There are no Type 6 definitions.

3.3.7 Type 7 Definitions

There are no Type 7 definitions.

3.3.8 Type 8 Definitions

There are no Type 8 definitions.

3.4 Abbreviations and symbols

NOTE Type 1 abbreviations and symbols shall apply to all the types except the same abbreviation or symbols are used with other types, in that case the latter shall have precedence.

3.4.1 Type 1 Abbreviations and symbols

AE	Application Entity
AE-I	Application Entity Invocation
AL	Application Layer
AP	Application Process
Ap_	Prefix for Data types defined for AP ASE
Ar_	Prefix for Data types defined for AR ASE
APDU	Application Protocol Data Unit
AR	Application Relationship
AREP	Application Relationship End Point
ASE	Application Service Element

ASN.1	Abstract Syntax Notation One
BCD	Binary Coded Decimal
BER	Basic Encoding Rule
cnf	confirmation primitive
DI_	Prefix for Data types defined for Data Link Layer types
DL	Data Link
DLC	Data Link Connection
DLCEP	Data Link Connection End Point
DLSAP	Data Link Service Access Point
DLSDU	Data Link Service Data Unit
Dt_	Prefix for Data types defined for Data Type ASE
Err	Error (used to indicate an APDU type)
Er_	Prefix for Error types defined
Ev_	Prefix for Data types defined for Event ASE
FAL	Fieldbus Application Layer
Fi_	Prefix for Data types defined for Function Invocation ASE
FIFO	First In First Out
Gn_	Prefix for Data Types defined for general use
ID	Identifier
IEC	International Electrotechnical Commission
in	input primitive
ind	indication primitive
ISO	International Organization for Standardization
LAS	Link Active Scheduler
Lr_	Prefix for Data types defined for Load Region ASE
lsb	least significant bit
Mn_	Prefix for Data Types defined for Management ASE
msb	most significant bit
out	output primitive
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PICS	Protocol Implementation Conformance Statement
QoS	Quality Of Service
Req	Request (used to indicate an APDU type)
req	request primitive
Rsp	Response (used to indicate an APDU type)
rsp	response primitive
SAP	Service Access Point
SDU	Service Data Unit
ToS	Type Of Service
Vr_	Prefix for Data types defined for Variable ASE

3.4.2 Type 2 Abbreviations and symbols

ASCII	American Standard Code for Information Interchange
CID	connection ID
CIP	The control and information protocol. CIP includes both connected and unconnected messaging.
DLL	Data Link Layer
PDU	protocol data unit
OSI	open systems interconnection (see ISO 7498)
Rcv	receive
Rx	receive
SDU	service data unit
SEM	state event matrix
STD	state transition diagram, used to describe object behaviour
Tx	transmit
Xmit	transmit
CM_API	actual packet interval
O2T or O⇒T	originator to target (connection parameters)
CM_RPI	requested packet interval
TUI	table unique identifier
T2O or T⇒O	target to originator (connection parameters)

3.4.3 Type 3 Abbreviations and symbols

Ack	Acknowledgement
Add	Address
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Identifier
AR	Application Relationship
AREP	Application Relationship End Point
ARL	Application Relationship List
ASE	Application Service Element
Cfg	Configuration
cnf	confirmation primitive

CREP	Communication Relationship End Point
CRL	Communication Relationship List
D_	Destination
DL	Data Link
DLM	Data Link-management
DLSAP	Data Link Service Access Point
DLSDU	Data Link Service Data Unit
DMPM	Data Link Mapping Protocol Machine
DMPMM1	Data Link Mapping Protocol Machine DP-Master (Class 1)
DMPMM2	Data Link Mapping Protocol Machine DP-Master (Class 2)
DMPMS	Data Link Mapping Protocol Machine DP-Slave
DSAP	Destination Service Access Point
FAL	Fieldbus Application Layer
FIFO	First In First Out
FSPMM1	FAL Service Protocol Machine DP-Master (Class 1)
FSPMM2	FAL Service Protocol Machine DP-Master (Class 2)
FSPMS	FAL Service Protocol Machine DP-Slave
HSA	Highest Station Address
ID	Identifier
IEC	International Electrotechnical Commission
ind	indication primitive
Inp	Input
ISO	International Organization For Standardization
L_sdu	Link Service Data Unit
lsb	least significant bit
msb	most significant bit
OSI	Open Systems Interconnection
Outp	Output
Param	Parameter
PDU	Protocol Data Unit
RD_	Read
Rem	Remote
Req	Request (used to indicate an APDU type)
req	request primitive
REQ	Request to indicate a Request PDU
RES	Response to indicate a Response PDU
RM	Resource Manager
RPL_UPD	Reply Update
Rsp	Response (used to indicate an APDU type)
rsp	response primitive
S_	Source
SAP	Service Access Point
SCL	Security Level
SD	Start Delimiter
SDN	Send Data with no Acknowledge
SDR	Station Delay Responder
Seq	Sequence
Src	Source
SRD	Send and Request Data with Reply
SSAP	Source Service Access Point
TR	Technical Report
USIF	User Interface
WD	Watchdog

3.4.4 Type 4 Abbreviations and symbols

There are no Type 4 abbreviations and symbols

3.4.5 Type 5 Abbreviations and symbols

There are no Type 5 abbreviations and symbols

3.4.6 Type 6 Abbreviations and symbols

There are no Type 6 abbreviations and symbols

3.4.7 Type 7 Abbreviations and symbols

A	Actuator
ACSE	Standard ISO 8649/ISO8650
AEI	Application Entity Implementation
API	Invocation of the Application Process
BER	ISO8825 Basic Encoding Rules
BT	Buffer Transfer - Layer 2
C	Controller
CC	Cell Controller

CADM	Computer Assisted Design & Manufacturing
CAPM	Computer Assisted Product Management
Cnf	Confirm
CS	Control System
CSD	Control System Device
DLSE	Data Link Service Element
Ind	Service Indication
MCS	Messaging Common Services
MMS	Standard ISO9506
MPS	Periodic/Aperiodic Manufacturing Services
MSG	Messaging Services - Layer 2
MAP3.0	ISO functional Profile
OS	Operator station
Rp	Service Response
Rq	Request
S	Sensor

3.4.8 Type 8 Abbreviations and symbols

There are no Type 8 abbreviations and symbols

3.5 Conventions

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. The protocol specification for each of the ASEs is defined in this standard.

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

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

3.5.1 Conventions for Type 1

3.5.1.1 Conventions for Class Definitions

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

3.5.1.2 Abstract Syntax Conventions

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

3.5.2 Conventions for Type 2

3.5.2.1 Attribute specification

Attributes are defined in an Attribute Table using the format and terms defined in Figure 1.

Attribute ID	Name	Data type	Semantics of values

Figure 1: Attribute table format and terms

The **Attribute ID** shall be a unique integer identification value assigned to an attribute. The valid ranges for Attribute IDs shall be as specified in 5.1.1.9.1.3. The Attribute ID shall identify the particular attribute being accessed.

Name shall refer to the name assigned to the attribute. An attribute may contain sub-elements, which may also have names, as is the case with the **STRUCT** of data type. The attribute name shall be the name which appears first, or at the top row in the name column (the row that contains the Attribute ID).

Every attribute shall be assigned a **Data type** which shall be either an elementary or a derived data type. The data types specified for the defined attributes shall be used in all implementations.

NOTE The elementary data types are defined in IEC 61158-5.

Semantics of values shall specify the meaning of the value(s) of the attribute. If this information needs more room than can fit in the table it will immediately follow the Class Attribute table. Included in the Class Attribute table will be an appropriate reference to this information.

If a Class Attribute is *optional*, then a default value or a special case processing method shall be defined such that the Client (Requester) can process the error message that occurs when accessing those objects that choose not to implement the class attribute.

3.5.2.2 Common Services

3.5.2.2.1 Service_PDU definitions

Each service has unique parameters for request and response. The service request and response parameters shall be defined using service Request/Response parameter tables as defined in Figure 2.

Name	Type	Semantics of values

Figure 2: Service request/response parameter

Name shall refer to the name given to the service request/response parameter.

Type shall specify the data type of the service request/response parameter.

Semantics of values shall specify the meaning of the values of the service request/response parameter, e.g. "the value is counts of microseconds."

3.5.2.2.2 Get_Attribute_All Response

3.5.2.2.2.1 General Definition

When the Get_Attribute_All common service is included in the list of supported System/Object Management services for an object class, then the **Get_Attribute_All** response shall be detailed for this class : the sequence or order of the data returned in the Service_ResponsePDU shall be specified. There are three ways in which the Get_Attribute_All Service_ResponsePDU may be specified:

- list the ordering of the attributes in the response message;
- specify the actual data array of the response message;
- combine the above two methods such that the actual data array also contains the attribute numbers.

Whichever method is used, the rules specified in Table 1 shall be adhered to when specifying the Get_Attribute_All Service_ResponsePDU of an object class for both the Class Attributes and the Instance Attributes.

Table 1: Get_Attribute_All response service rules

Rule number	Rule
1	If the definition of the Get_Attribute_All response includes optional attributes, then default values shall be specified in the response description. These defaults shall be used if an implementation does not support the optional attributes. If any of the following optional attributes are included in an object specification, but not supported in the implementation of the object, then the following shall be adhered to: - if the class attribute "Optional attribute list" is not supported, the default value of zero shall be inserted into the response array and no optional attribute numbers shall follow; - if the class attribute "optional service list" is not supported, the default value of zero shall be inserted into the response array and no optional service numbers shall follow.
2	If new attributes are added to an existing object, those attributes shall be added to the end of the response attribute list or data array to ensure compatibility with different object revisions.
3	Whichever method is used to specify the response, it shall be done in such a way as to be unambiguous, including rules to deal with variable length fields and padding.
4	The Get_Attribute_All response shall include only the open attributes; it shall not include any vendor specific attributes.

Table 2 is an example of the attribute ordering method of specifying the service data portion of a Get_Attribute_All response for class level attributes of an object which supports **optional gettable** class attributes 1,2,3 and 4.

Table 2: Example Class level object/service specific reply data of Get_Attribute_All

Class attribute ID	Attribute name and default value
1	Revision, default = 0x0001
2	Max Instance, default = 0x0000
3	Number of Instances, default = 0x0000
4	Attribute list, number of attributes, default = 0x0000

Table 3 is an example of the data array method of specifying the service data portion of a Get_Attribute_All response for class level attributes of an object which supports **optional gettable** class attributes 1,2,3 and 4.

Table 3: Example Get_Attribute_All data array method

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Revision (low byte) Default = 1							
1	Revision (high byte) Default = 0							
2	Max Instance (low byte) Default = 0							
3	Max Instance (high byte) Default = 0							
4	Number of Instances (low byte) Default = 0							
5	Number of Instances (high byte) Default = 0							
6	Optional Attribute List : number of attributes (low byte) Default = 0							
7	Optional Attribute List : number of attributes (high byte) Default = 0							
8	Optional Attribute List : optional attribute #1 (low byte)							
9	Optional Attribute List : optional attribute #1 (high byte)							
2n + 6	Optional Attribute List : optional attribute #n (low byte)							
2n + 7	Optional Attribute List : optional attribute #n (high byte)							

3.5.2.2.2 Revisions

The defined Get_Attribute_All response for an object may increase in size with each revision of the object; however, to insure interoperability, the format of the first part of the response shall remain parseable by a client of the older revision of the object. Clients (Requesters) need not make use of this compatibility requirement.

NOTE The Revision class attribute is not the revision of an implementation (which is reflected in the Identity Object, Minor/Major revision status bits), but the revision of the *class definition*.

3.5.2.2.3 Set_Attribute_All Request

3.5.2.2.3.1 General Definition

When the Set_Attribute_All common service is included in the list of supported common services, then the format of the **Set_Attribute_All request** shall be included in the object specification. The sequence or order of the data supplied in the Service Data portion of the request shall be specified. There are three ways in which the Set_Attribute_All request may be specified:

- list the order of the attributes in the request message;
- specify the actual data array of the request message;
- combine the above two methods such that the actual data array also contains the attribute numbers.

Whichever method is used, the rules specified in Table 4 shall be adhered to when specifying an object's Set_Attribute_All request in the object specification for both the Class Attributes and the Instance Attributes.

Table 4: Set_Attribute_All request service rules

Rule number	Rule
1	An object shall support the Set_Attribute_All service only if all settable attributes shown in the Set_Attribute_All request are implemented as settable.
2	If new settable attributes are added to an existing object, those attributes shall be added to the end of the request attribute list or data array.
3	Whichever method is used to specify the response, it shall be done in such a way as to be unambiguous, including rules to deal with variable length fields and padding.
4	The Set_Attribute_All request shall include only the open attributes. It shall not include any vendor specific attributes.

Table 5 is an example of the attribute ordering method of specifying the service data portion of a Set_Attribute_All request for instance level attributes of an object which supports **required** **settable** instance attributes 7,8,9,10,11 and 12.

Table 5: Example Set Attribute All attribute ordering method

Class Attribute ID	Attribute name
7	Output Range
8	Value Data Type
9	Fault State
10	Idle State
11	Fault Value
12	Idle Value

Table 6 is an example of the data array method of specifying the service data portion of a Set_Attribute_All request for instance level attributes of an object which supports **required settable** instance attributes 7,8,9,10,11 and 12.

Table 6: Example Set Attribute All data array method

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Output Range							
1	Value Data Type							
2	Fault State							
3	Idle State							
4	Fault Value (low byte)							
5	Fault Value (high byte)							
6	Idle Value (low byte)							
7	Idle Value (high byte)							

3.5.2.3.2 Revisions

A server processing a Set_Attribute_All request may need to process more data than is expected by the server if the client has implemented a newer revision of this object while the server an older revision. As a result, the server object may have to process a Set_Attribute_All request that contains more data because the additional attributes were not recognised. If the server receives more data in a Set_Attribute_All request than it expects the server shall respond with a general status code equal to *Too much data* (0x15).

NOTE The Revision class attribute is not the revision of an implementation (which is reflected in the Identity Object, Minor/Major revision status bits), but the revision of the *class definition*.

3.5.3 Conventions for Type 3

3.5.3.1 Abstract Syntax Conventions

The following conventions and rules are used:

1. PDUs are described as octets or groups of octets
2. Groups of octets separated by a comma appear in the order they are transferred
4. If optional octets are not present the following octets appear without a gap
5. If octets or groups of octets are grouped within "{ }" the order is arbitrary
6. If octets or groups of octets are marked with "*" they may appear more than once. If it is used within a "{ }" section the may appear mixed with other octets or group of octets of this section.
7. Octets can be grouped or values can be assigned within "()"
8. Complex APDUs may be built by means of substitutions (sub-structures)
9. Exclusive selections of octets or groups of octets are separated by "^"

NOTE1 The formal PDU example

APOE_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

NOTE2 The arbitrary order implies that groups of octets are characterised by a special header that is described within the coding rules.

NOTE3 The APDU syntax implies that according to the maximum DLSDU a APDU shall not exceed 244 octets in total.

3.5.3.2 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.

3.5.3.3 Conventions for the Common Coding Structure 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 following order.

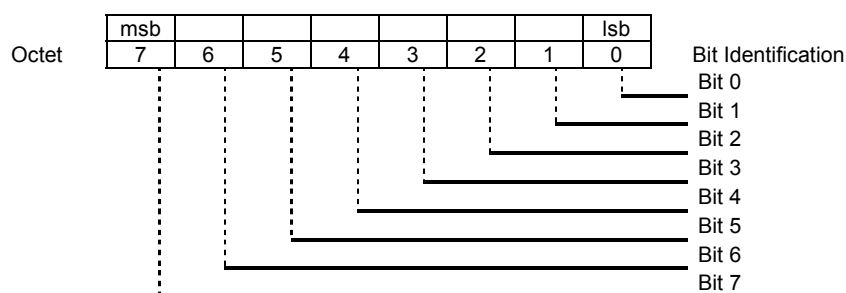


Figure 3 - Common structure of specific Fields

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 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 bits.

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.

3.5.4 Conventions for Type 4

There are no Type 4 specific conventions defined.

3.5.5 Conventions for Type 5

There are no Type 5 specific conventions defined.

3.5.6 Conventions for Type 6

There are no Type 6 specific conventions defined.

3.5.7 Conventions for Type 7

There are no Type 7 specific conventions defined.

3.5.8 Conventions for Type 8

There are no Type 8 specific conventions defined.

3.6 Conventions used in State Machines

3.6.1 Conventions for Type 1

The State Machines are described as follows:

Table 7 – Conventions used for State Machines

No.	Current State	Events Actions	Next state
Name of this transition.	The current state to which this state transition applies.	Events or conditions that trigger this state transaction. The actions that are taken when the above events or conditions are met. The actions are always indented below events or conditions.	The next state after the actions in this transition is taken.

The conventions used in the state machines are as follows:

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

xxx A parameter name.

Example:

Identifier := reason

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

"xxx" Indicates fixed value.

Example:

Identifier := "abc"

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

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

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

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

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

&& Logical "AND"

|| Logical "OR"

```
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
endif
```

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.

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

3.6.2 Conventions for Type 2

3.6.2.1 State Machine Conventions

State changes may be triggered by events (internal or external) or service invocations. Reaction to service invocations may depend on the value(s) of the attribute(s) accessed by the service.

Behaviour of the state machine shall be defined for combinations of :

- the *event* the machine receives;
- the *state* the machine is in when it receives notification of a state-changing event.

To define behaviour in these terms, a State Transition Diagram (STD) and a State Event Matrix (SEM) are used when applicable in the state machine specification.

3.6.2.1.1 State Transition Diagram

An **event** is an external stimulus that may cause a state transition. A **State Transition Diagram (STD)** graphically illustrates the states of an object and includes events, service calls and changes of attributes that cause it to transition to another state. Figure 4 shows an example of a State Transition Diagram.

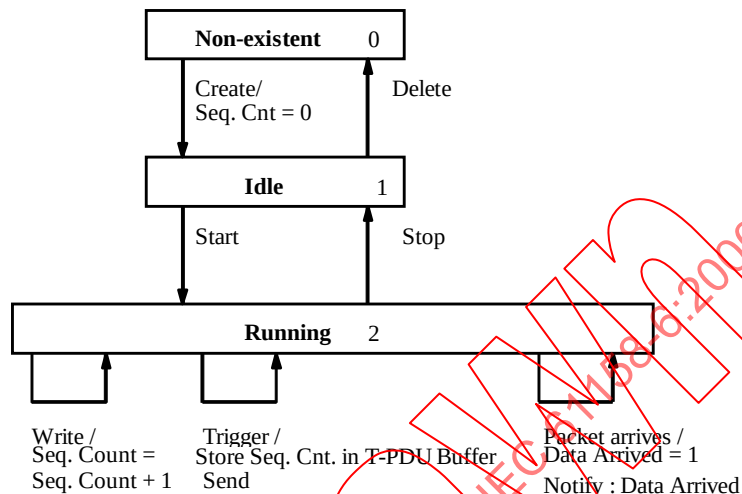


Figure 4: Example of State Transition Diagram (STD)

NOTE Event that is the primary cause of the State Transition is followed by “/” mark. Notification to upper layer is marked by “Notify :”

3.6.2.1.2 State Event Matrix

A **state** is the current active mode of operation of the state machine object (e.g. Running, Idle).

A **State Event Matrix** is a table that lists all possible events, services and changes in attribute values that initiate a state change, and indicates the response by the object to the event based on the state of the object when it receives notification of that event.

State Event Matrix format is as shown in Table 8:

Table 8: State Event Matrix Format

Event	State		
	State 1		State n
Event A description	Function triggered by event A in State 1 (if any) Notification to FAL user (if any) Transition to another state (if any)		Function triggered by event A in State n (if any) Notification to FAL user (if any) Transition to another state (if any)
.....			
Event X description	Function triggered by event X in State 1 (if any) Notification to FAL user (if any) Transition to another state (if any)		Function triggered by event X in State n (if any) Notification to FAL user (if any) Transition to another state (if any)

NOTE In absence of function, notification or transition, the corresponding entry shall not be made in the table.

3.6.2.1.3 Example State Event Matrix (Informative)

Table 9 shows an example of a state machine with three states :

- **Non-existent:** the object has not yet been created; objects transition to the existent state via the create service (if the object may be dynamically created) or at power-up (if the object is fixed by design/implementation);
- **Idle:** the object accepts services (e.g. Get_Attribute_Single), but does not produce or consume data onto or from the link;
- **Running:** the object is performing all its specified functions.

Table 9: Example State Event Matrix

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error: Object already exists.	Error: Object already exists.
Delete	Error: Object does not exist. (General Error Code 0x16)	Transition to Non-existent	Error: Object State Conflict (General Error Code 0x0C)
Start	Error: Object does not exist. (General Error Code 0x16)	Transition to Running	Error: Object State Conflict (General Error Code 0x0C)
Stop	Error: Object does not exist. (General Error Code 0x16)	Error: Object State Conflict (General Error Code 0x0C)	Transition to Idle
Get_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Validate/service the request. Return response
Set_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Error: Object State Conflict (General Error Code 0x0C)

3.6.3 Conventions for Type 3

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:

- The first row contains the name of the transition.
- The second row in capital letters define the current state.
- The third row contains an optional event in bold characters,
Conditions start with a “\” as first line character followed by the conditions and
Actions start with a “=>” as first line character followed by the action code.
- The last row the next state.

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

Layout of a Machine description:

TRAN- SITION	CURRENT STATE	Event \ Condition => Action	NEXT STATE

Table 10 - State Machine Description elements

Description element	Meaning
Current State	Name of the given states.
Next State	
Transition	Name 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 as follows:

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

xxx A parameter name.

Example:

Identifier := reason

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

"xxx" Indicates fixed value.

Example:

Identifier := "abc"

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

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

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

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

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

&& Logical "AND"

|| Logical "OR"

```
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
```

```
endif
```

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.

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)

3.6.4 Conventions for Type 4

There are no Type 4 specific conventions defined.

3.6.5 Conventions for Type 5

There are no Type 5 specific conventions defined.

3.6.6 Conventions for Type 6

There are no Type 6 specific conventions defined.

3.6.7 Conventions for Type 7

This section recalls the main definitions associated with the three specification techniques used : 'Evaluation Networks', 'Abstract objects', 'Service Conventions'.

3.6.7.1 Evaluation networks

To be helpful for the reader some mechanisms are explained by a graphical representation. These representations are based on evaluation network built on an oriented graph with three types of nodes:

- . The states, represented by a disk,
- . The requests represented by a rectangle,
- . The transitions, represented by a horizontal line.

The evaluation network is built according to the following principle:

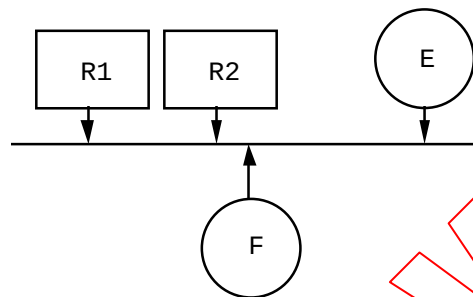


Figure 5: Example of an evaluation net

When the described system is in state E, the arrival of request R1 or R2 results in a transition which switches it state F.

On the other hand, crossing a transition takes place, by convention, in zero time.

3.6.7.2 Simple actions

Simple actions can be associated with each transition, corresponding either to the transmission of requests or to the execution of local actions.

They are materially represented by a triangle, labelled with the name of the request or of the description of the executed action.

3.6.7.3 Conditions

The crossing of a transition can concern a condition. In this case, the oriented graph associated with the representation, shows a condition associated with the arc which precedes the transition.

A corollary of this representation allows defining choices for crossing a transition from the same initial state.

In the same way as for a simple transition, the transmission of an action can be associated with the crossing of a conditional transition.

3.6.8 Conventions for Type 8

There are no Type 8 specific conventions defined.

4 Type 1

4.1 FAL Syntax Description

FalArHeader

```
FalArHeader ::= Unsigned8 {
    -- bit 8      FAL Protocol Specifier      (Always 0 for the protocol defined by this part of IEC 61158-2)
    -- bit 7 - 4  Protocol Identifier          (Identifies abstract syntax, revision, and encoding rules)
    -- bit 3      Protocol Specific bit       (Reserved for each protocol to use)
    -- bit 2-1    PDU Identifier              (Identifies a PDU type within a Protocol Identifier)
}
```

The FalArHeader shall be the first octet of all the FAL PDUs. It enables multiple protocols as well as encoding rules to coexist on the same network. subclause 4.2.1.1 defines the code points for the currently identified combinations of protocol and encoding rules.

4.1.1 FAL-AR PDU Abstract Syntax 1

The productions defined here shall be used with the Compact Encoding Rules, and it is strongly recommended that they be used with the messaging and buffered encoding rules to produce the shortest PDUs that are suitable for a limited bandwidth underlying layer.

```
FalArPDU ::= ConfirmedSend-CommandPDU
            || ConfirmedSend-AffirmativePDU
            || ConfirmedSend-NegativePDU
            || UnconfirmedSend-CommandPDU
            || UnconfirmedAcknowledgedSend-CommandPDU
            || UnconfirmedAcknowledgedSend-AffirmativePDU
            || Idle-CommandPDU
            || AR-XON-OFF-CommandPDU
            || Establish-CommandPDU
            || Establish-AffirmativePDU
            || Establish-NegativePDU
            || Abort-PDU
```

4.1.1.1 Confirmed Send Service

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

ConfirmedSend-AffirmativePDU ::= SEQUENCE {
    FalArHeader,
    InvokeID,
    ConfirmedServiceResponse
}

ConfirmedSend-NegativePDU ::= SEQUENCE {
    FalArHeader,
    InvokeID,
    ConfirmedServiceError
}
```

4.1.1.2 Unconfirmed Send Service

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

4.1.1.3 Unconfirmed Acknowledged Send Service

```
UnconfirmedAcknowledgedSend-CommandPDU ::= SEQUENCE {
    FalArHeader,
    unconfirmed-PDU [4] IMPLICIT SEQUENCE {
        InvokeID,
        UnconfirmedServiceRequest
    }
}
```

```

UnconfirmedAcknowledgedSend-AffirmativePDU ::= {
    FalArHeader
}

```

4.1.1.4 Idle Send Service

```

IdleSend-CommandPDU ::= {
    FalArHeader
}

```

4.1.1.5 AR-XON-OFF Send Service

```

AR-XON-OFF-CommandPDU ::= SEQUENCE {
    FalArHeader,
    AR-XON-OFFPDU
    Invoke ID,
    XON-OFF
}

```

[7] IMPLICIT SEQUENCE {

4.1.1.6 Establish Service

```

Establish-CommandPDU ::= SEQUENCE {
    FalArHeader,
    RespondingAREP,
    RequestingAREP,
    MAXOSCC,
    MAXOSCS,
    MAXUCSC,
    MAXUCSS,
    CIU,
    InvokeID,
    Initiate-RequestPDU
}

```

```

Establish-AffirmativePDU ::= SEQUENCE {
    FalArHeader,
    InvokeID,
    Initiate-ResponsePDU
}

```

```

Establish-NegativePDU ::= SEQUENCE {
    FalArHeader,
    InvokeID,
    Initiate-ErrorPDU
}

```

4.1.2 FAL-AR PDU Abstract Syntax 2

The productions defined here shall be used with the Traditional Encoding Rules to maintain compatibility with prior standards.

```

FalArPDU ::= ConfirmedSend-RequestPDU
|| ConfirmedSend-ResponsePDU
|| UnconfirmedSend-PDU
|| UnconfirmedAcknowledgedSend-RequestPDU
|| UnconfirmedAcknowledgedSend-ResponsePDU
|| Idle-CommandPDU
|| AR-XON-OFF-CommandPDU
|| Establish-RequestPDU
|| Establish-Request2PDU
|| Establish-ResponsePDU
|| Establish-ErrorPDU
|| Abort-PDU

```

4.1.2.1 Confirmed Send Service

```

ConfirmedSend-RequestPDU ::= SEQUENCE {
    FalArHeader,
    confirmed-RequestPDU
    InvokeID,
    ConfirmedServiceRequest
}

```

[1] IMPLICIT SEQUENCE {

```
ConfirmedSend-ResponsePDU ::= SEQUENCE {
    FalArHeader,
    pduBody CHOICE {
        confirmed-ResponsePDU [2] IMPLICIT SEQUENCE {
            InvokeID,
            ConfirmedServiceResponse
        },
        confirmed-ErrorPDU [3] IMPLICIT SEQUENCE {
            InvokeID,
            ConfirmedServiceError
        }
    }
}
```

4.1.2.2 Unconfirmed Send Service

```
UnconfirmedSend-PDU ::= SEQUENCE {
    FalArHeader,
    unconfirmed-PDU [4] IMPLICIT SEQUENCE {
        InvokeID,
        UnconfirmedServiceRequest
    }
}
```

4.1.2.3 Unconfirmed Acknowledged Send Service

```
UnconfirmedAcknowledgedSend-RequestPDU ::= SEQUENCE {
    FalArHeader,
    unconfirmed-PDU [4] IMPLICIT SEQUENCE {
        InvokeID,
        UnconfirmedServiceRequest
    }
}
```

```
UnconfirmedAcknowledgedSend-ResponsePDU ::= {
    FalArHeader
}
```

4.1.2.4 Idle Send Service

```
IdleSend-CommandPDU ::= {
    FalArHeader
}
```

4.1.2.5 AR-XON-OFF Send Service

```
AR-XON-OFF-CommandPDU ::= SEQUENCE {
    FalArHeader,
    AR-XON-OFFPDU [7] IMPLICIT SEQUENCE {
        Invoke ID,
        XON-OFF
    }
}
```

4.1.2.6 Establish Service

```
Establish-RequestPDU ::= SEQUENCE {
    FalArHeader,
    RespondingAREP,
    RequestingAREP,
    userData [6] IMPLICIT SEQUENCE {
        InvokeID,
        InitiateRequest [0] IMPLICIT Initiate-RequestPDU
    }
}
```

```

Establish-Request2PDU ::= SEQUENCE {
    FalArHeader,
    RespondingAREP,
    RequestingAREP,
    AREPContext
    MaxOSCC,
    MaxOSCS,
    MAXUCSC,
    MAXUCSS,
    CIU
}

```

[5] IMPLICIT SEQUENCE {

```

Establish-ResponsePDU ::= SEQUENCE {
    FalArHeader,
    userData
    InvokeID,
    InitiateResponse
}

```

[6] IMPLICIT SEQUENCE {

[1] IMPLICIT Initiate-ResponsePDU

```

Establish-ErrorPDU ::= SEQUENCE {
    FalArHeader,
    userData
    InvokeID,
    initiateError
}

```

[6] IMPLICIT SEQUENCE {

[2] IMPLICIT Initiate-ErrorPDU

4.1.2.7 MaxOSCC

MaxOSCC ::= Unsigned8

4.1.2.8 MaxOSCS

MaxOSCS ::= Unsigned8

4.1.2.9 MaxUCSC

MaxUCC ::= Unsigned8

4.1.2.10 MaxUCSS

MaxUCS ::= Unsigned8

4.1.2.11 XON_OFF

XON_OFF ::= Unsigned8

4.1.2.12 CIU

CIU ::= Unsigned32

4.1.3 Abstract Syntax of PDUBody

4.1.3.1 Abort Service

```

Abort-PDU ::= SEQUENCE {
    FalArHeader,
    Identifier,
    ReasonCode,
    AdditionalDetail
}

```

4.1.3.2 InvokeID

InvokeID ::= Unsigned8

4.1.3.3 ConfirmedServiceRequest

ConfirmedServiceRequest ::= CHOICE {

status-Request	[0] IMPLICIT Status-RequestPDU,
identify-Request	[1] IMPLICIT Identify-RequestPDU,
read2-Request	[2] IMPLICIT Read2-RequestPDU,
write-Request	[3] IMPLICIT Write-RequestPDU,
getAttributes2-Request	[4] IMPLICIT GetAttributes2-RequestPDU,
readWithType-Request	[5] IMPLICIT ReadWithType-RequestPDU,
writeWithType-Request	[6] IMPLICIT WriteWithType-RequestPDU,
defineVariableList-Request	[7] IMPLICIT DefineVariableList-RequestPDU,
deleteVariableList-Request	[8] IMPLICIT DeleteVariableList-RequestPDU,
initiateClientPullDownloadStatic-Request	[9] IMPLICIT InitiateClientPullDownloadStatic-RequestPDU,
pullDownloadSegment-Request	[10] IMPLICIT PullDownloadSegment-RequestPDU,
terminatePullDownload-Request	[11] IMPLICIT TerminatePullDownload-RequestPDU,
initiateClientPullUploadStatic-Request	[12] IMPLICIT InitiateClientPullUploadStatic-RequestPDU,
pullUploadSegment-Request	[13] IMPLICIT PullUploadSegment-RequestPDU,
terminatePullUpload-Request	[14] IMPLICIT TerminatePullUpload-RequestPDU,
initiateServerPullDownload-Request	[15] IMPLICIT InitiateServerPullDownload-RequestPDU,
initiateServerPullUpload-Request	[16] IMPLICIT InitiateServerPullUpload-RequestPDU,
createFunctionInvocation-Request	[17] IMPLICIT CreateFunctionInvocation-RequestPDU,
deleteFunctionInvocation-Request	[18] IMPLICIT DeleteFunctionInvocation-RequestPDU,
start-Request	[19] IMPLICIT Start-RequestPDU,
stop-Request	[20] IMPLICIT Stop-RequestPDU,
resume-Request	[21] IMPLICIT Resume-RequestPDU,
reset-Request	[22] IMPLICIT Reset-RequestPDU,
kill-Request	[23] IMPLICIT Kill-RequestPDU,
enableEvent-Request	[24] IMPLICIT EnableEvent-RequestPDU,
acknowledgeEventNotification-Request	[25] IMPLICIT AcknowledgeEventNotification-RequestPDU,
physicalRead-Request	[26] IMPLICIT PhysicalRead-RequestPDU,
physicalWrite-Request	[27] IMPLICIT PhysicalWrite-RequestPDU,
beginSetAttributes-Request	[28] IMPLICIT BeginSetAttributes-RequestPDU,
setAttributes2-Request	[29] IMPLICIT SetAttributes2-RequestPDU,
endSetAttributes-Request	[30] IMPLICIT EndSetAttributes-RequestPDU,
initiateClientPushDownload-Request	[31] IMPLICIT InitiateClientPushDownload-RequestPDU,
pushDownloadSegment-Request	[32] IMPLICIT PushDownloadSegment-RequestPDU,
terminatePushDownload-Request	[33] IMPLICIT TerminatePushDownload-RequestPDU,
discard-Request	[34] IMPLICIT Discard-RequestPDU,
readList-Request	[35] IMPLICIT ReadList-RequestPDU,
writeList-Request	[36] IMPLICIT WriteList-RequestPDU,
exchange-Request	[37] IMPLICIT Exchange-RequestPDU,
getEventSummary-Request	[38] IMPLICIT GetEventSummary-RequestPDU,
getEventSummaryList-Request	[39] IMPLICIT GetEventSummaryList-RequestPDU,
setAttributes1-Request	[40] IMPLICIT SetAttributes1-RequestPDU,
getAttributes1-Request	[41] IMPLICIT GetAttributes1-RequestPDU,
read1-Request	[42] IMPLICIT Read1-RequestPDU,
conclude-Request	[43] IMPLICIT Conclude-RequestPDU,
subscribe-Request	[44] IMPLICIT Subscribe-RequestPDU,
initiateClientPullDownloadDynamic-Request	[45] IMPLICIT InitiateClientPullDownloadDynamic-RequestPDU,
initiateClientPullUploadDynamic-Request	[46] IMPLICIT InitiateClientPullUploadDynamic-RequestPDU,
exchangeList-Request	[47] IMPLICIT ExchangeList-RequestPDU,
enableEventList-Request	[48] IMPLICIT EnableEventList-RequestPDU,
confirmedAcknowledgeEventList-Request	[49] IMPLICIT ConfirmedAcknowledgeEventList-RequestPDU,
getAttributes3-Request	[50] IMPLICIT GetAttributes3-RequestPDU,
queryEventSummaryList-Request	[51] IMPLICIT QueryEventSummaryList-RequestPDU

}

4.1.3.4 ConfirmedServiceResponse

ConfirmedServiceResponse ::= CHOICE {

status-Response
 identify-Response
 read2-Response
 write-Response
 getAttributes2-Response
 readWithType-Response
 writeWithType-Response
 defineVariableList-Response
 deleteVariableList-Response
 initiateClientPullDownloadStatic-Response
 pullDownloadSegment-Response
 terminatePullDownload-Response
 initiateClientPullUploadStatic-Response
 pullUploadSegment-Response
 terminatePullUpload-Response
 initiateServerPullDownload-Response
 initiateServerPullUpload-Response
 createFunctionInvocation-Response
 deleteFunctionInvocation-Response
 start-Response
 stop-Response
 resume-Response
 reset-Response
 kill-Response
 enableEvent-Response
 acknowledgeEventNotification-Response
 physicalRead-Response
 physicalWrite-Response
 beginSetAttributes-Response
 setAttributes2-Response
 endSetAttributes-Response
 initiateClientPushDownload-Response
 pushDownloadSegment-Response
 terminatePushDownload-Response
 discard-Response
 readList-Response
 writeList-Response
 exchange-Response
 getEventSummary-Response
 getEventSummaryList-Response
 setAttributes1-Response
 getAttributes1-Response
 read1-Response
 conclude-Response
 subscribe-Response
 initiateClientPullDownloadDynamic-Response
 initiateClientPullUploadDynamic-Response
 exchangeList-Response
 enableEventList-Response
 confirmedAcknowledgeEventList-Response
 getAttributes3-Response
 queryEventSummaryList-Response

[0] IMPLICIT Status-ResponsePDU,
 [1] IMPLICIT Identify-ResponsePDU,
 [2] IMPLICIT Read2-ResponsePDU,
 [3] IMPLICIT Write-ResponsePDU,
 [4] IMPLICIT GetAttributes2-ResponsePDU,
 [5] IMPLICIT ReadWithType-ResponsePDU,
 [6] IMPLICIT WriteWithType-ResponsePDU,
 [7] IMPLICIT DefineVariableList-ResponsePDU,
 [8] IMPLICIT DeleteVariableList-ResponsePDU,
 [9] IMPLICIT InitiateClientPullDownloadStatic-ResponsePDU,
 [10] IMPLICIT PullDownloadSegment-ResponsePDU,
 [11] IMPLICIT TerminatePullDownload-ResponsePDU,
 [12] IMPLICIT InitiateClientPullUploadStatic-ResponsePDU,
 [13] IMPLICIT PullUploadSegment-ResponsePDU,
 [14] IMPLICIT TerminatePullUpload-ResponsePDU,
 [15] IMPLICIT InitiateServerPullDownload-ResponsePDU,
 [16] IMPLICIT InitiateServerPullUpload-ResponsePDU,
 [17] IMPLICIT CreateFunctionInvocation-ResponsePDU,
 [18] IMPLICIT DeleteFunctionInvocation-ResponsePDU,
 [19] IMPLICIT Start-ResponsePDU,
 [20] IMPLICIT Stop-ResponsePDU,
 [21] IMPLICIT Resume-ResponsePDU,
 [22] IMPLICIT Reset-ResponsePDU,
 [23] IMPLICIT Kill-ResponsePDU,
 [24] IMPLICIT EnableEvent-ResponsePDU,
 [25] IMPLICIT AcknowledgeEventNotification-ResponsePDU,
 [26] IMPLICIT PhysicalRead-ResponsePDU,
 [27] IMPLICIT PhysicalWrite-ResponsePDU,
 [28] IMPLICIT BeginSetAttributes-ResponsePDU,
 [29] IMPLICIT SetAttributes2-ResponsePDU,
 [30] IMPLICIT EndSetAttributes-ResponsePDU,
 [31] IMPLICIT InitiateClientPushDownload-ResponsePDU,
 [32] IMPLICIT PushDownloadSegment-ResponsePDU,
 [33] IMPLICIT TerminatePushDownload-ResponsePDU,
 [34] IMPLICIT Discard-ResponsePDU,
 [35] IMPLICIT ReadList-ResponsePDU,
 [36] IMPLICIT WriteList-ResponsePDU,
 [37] IMPLICIT Exchange-ResponsePDU,
 [38] IMPLICIT GetEventSummary-ResponsePDU,
 [39] IMPLICIT GetEventSummaryList-ResponsePDU,
 [40] IMPLICIT SetAttributes1-ResponsePDU,
 [41] IMPLICIT GetAttributes1-ResponsePDU,
 [42] IMPLICIT Read1-ResponsePDU,
 [43] IMPLICIT Conclude-ResponsePDU,
 [44] IMPLICIT Subscribe-ResponsePDU,
 [45] IMPLICIT InitiateClientPullDownloadDynamic-ResponsePDU,
 [46] IMPLICIT InitiateClientPullUploadDynamic-ResponsePDU,
 [47] IMPLICIT ExchangeList-ResponsePDU,
 [48] IMPLICIT EnableEventList-ResponsePDU,
 [49] IMPLICIT ConfirmedAcknowledgeEventList-ResponsePDU,
 [50] IMPLICIT GetAttributes3-ResponsePDU,
 [51] IMPLICIT QueryEventSummaryList-ResponsePDU

}

4.1.3.5 ConfirmedServiceError

```
ConfirmedServiceError ::= CHOICE {
    status-Error           [0] IMPLICIT ErrorType,
    identify-Error         [1] IMPLICIT ErrorType,
    read2-Error            [2] IMPLICIT ErrorType,
    write-Error            [3] IMPLICIT ErrorType,
    getAttributes2-Error   [4] IMPLICIT ErrorType,
    readWithType-Error     [5] IMPLICIT ErrorType,
    writeWithType-Error    [6] IMPLICIT ErrorType,
    defineVariableList-Error [7] IMPLICIT ErrorType,
    deleteVariableList-Error [8] IMPLICIT ErrorType,
    initiateClientPullDownloadStatic-Error [9] IMPLICIT ErrorType,
    pullDownloadSegment-Error [10] IMPLICIT ErrorType,
    terminatePullDownload-Error [11] IMPLICIT ErrorType,
    initiateClientPullUploadStatic-Error [12] IMPLICIT ErrorType,
    pullUploadSegment-Error [13] IMPLICIT ErrorType,
    terminatePullUpload-Error [14] IMPLICIT ErrorType,
    initiateServerPullDownload-Error [15] IMPLICIT ErrorType,
    initiateServerPullUpload-Error [16] IMPLICIT ErrorType,
    createFunctionInvocation-Error [17] IMPLICIT ErrorType,
    deleteFunctionInvocation-Error [18] IMPLICIT ErrorType,
    start-Error           [19] IMPLICIT ErrorType,
    stop-Error            [20] IMPLICIT ErrorType,
    resume-Error          [21] IMPLICIT ErrorType,
    reset-Error           [22] IMPLICIT ErrorType,
    kill-Error            [23] IMPLICIT ErrorType,
    enableEvent-Error     [24] IMPLICIT ErrorType,
    acknowledgeEventNotification-Error [25] IMPLICIT ErrorType,
    physicalRead-Error    [26] IMPLICIT ErrorType,
    physicalWrite-Error   [27] IMPLICIT ErrorType,
    beginSetAttributes-Error [28] IMPLICIT ErrorType,
    setAttributes2-Error  [29] IMPLICIT ErrorType,
    endSetAttributes-Error [30] IMPLICIT ErrorType,
    initiateClientPushDownload-Error [31] IMPLICIT ErrorType,
    pushDownloadSegment-Error [32] IMPLICIT ErrorType,
    terminatePushDownload-Error [33] IMPLICIT ErrorType,
    discard-Error         [34] IMPLICIT ErrorType,
    readList-Error        [35] IMPLICIT ErrorType,
    writeList-Error       [36] IMPLICIT ErrorType,
    exchange-Error        [37] IMPLICIT ErrorType,
    getEventSummary-Error [38] IMPLICIT ErrorType,
    getEventSummaryList-Error [39] IMPLICIT ErrorType,
    setAttributes1-Error  [40] IMPLICIT ErrorType,
    getAttributes1-Error  [41] IMPLICIT ErrorType,
    read1-Error           [42] IMPLICIT ErrorType,
    conclude-Error        [43] IMPLICIT ErrorType,
    subscribe-Error       [44] IMPLICIT ErrorType,
    initiateClientPullDownloadDynamic-Error [45] IMPLICIT ErrorType,
    initiateClientPullDownloadDynamic-Error [46] IMPLICIT ErrorType,
    exchangeList-Error    [47] IMPLICIT ErrorType,
    enableEventList-Error [48] IMPLICIT ErrorType,
    confirmedAcknowledgeEventList-Error [49] IMPLICIT ErrorType,
    getAttributes3-Error  [50] IMPLICIT ErrorType,
    queryEventSummaryList-Error [51] IMPLICIT ErrorType,
    xon-off-Error         [52] IMPLICIT ErrorType
}
```

4.1.3.6 Error Type

```
ErrorType ::= SEQUENCE {
    errorClass           [0] IMPLICIT ErrorClass,
    optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    additionalCode       [1] IMPLICIT Integer16 OPTIONAL,
    additionalDescription [2] IMPLICIT VisibleString OPTIONAL,
    serviceSpecificInfo  [3] IMPLICIT ANY OPTIONAL
}
```

4.1.3.7 Error Class

```

ErrorClass ::= CHOICE {
    vfdState
        other
    },
    applicationReference
        other
        application-unreachable
        application-reference-invalid
        context-unsupported
    },
    definition
        other
        object-undefined
        object-attributes-inconsistent
        name-already-exists
        type-unsupported
        type-inconsistent
    },
    resource
        other
        memory-unavailable
    },
    service
        other
        object-state-conflict
        pdu-size
        object-constraint-conflict
        parameter-inconsistent
        illegal-parameter
    },
    access
        other
        object-invalidated
        hardware-fault
        object-access-denied
        invalid-address
        object-attribute-inconsistent
        object-access-unsupported
        object-non-existent
        type-conflict
        named-access-unsupported
        access-to-element-unsupported
    },
    objectDescription
        other
        name-length-overflow
        od-overflow
        od-write-protected
        extension-length-overflow
        od-description-length-overflow
        operational-problem
    },
    conclude
        other
        further-communication-required
    },
    other
}

```

[1] IMPLICIT Integer8 {
(0)

[2] IMPLICIT Integer8 {
(0),
(1),
(2),
(3)

[3] IMPLICIT Integer8 {
(0),
(1),
(2),
(3),
(4),
(5)

[4] IMPLICIT Integer8 {
(0),
(1)

[5] IMPLICIT Integer8 {
(0),
(1),
(2),
(3),
(4),
(5)

[6] IMPLICIT Integer8 {
(0),
(1),
(2),
(3),
(4),
(5),
(6),
(7),
(8),
(9),
(10)

[7] IMPLICIT Integer8 {
(0),
(1),
(2),
(3),
(4),
(5),
(6)

[9] IMPLICIT Integer8 {
(0),
(1)

[8] IMPLICIT Integer8 {
(0)

4.1.3.8 Unconfirmed PDUs

```
UnconfirmedServiceRequest ::= CHOICE {
    informationReport-Request          [0] IMPLICIT InformationReport-RequestPDU,
    unsolicitedStatus-Request          [1] IMPLICIT UnsolicitedStatus-RequestPDU,
    eventNotification-Request          [2] IMPLICIT EventNotification-RequestPDU,
    informationReportWithType-Request [3] InformationReportWithType-RequestPDU,
    eventNotificationWithType-Request [4] EventNotificationWithType-RequestPDU,
    actionInvoke-Request               [5] IMPLICIT ActionInvoke-RequestPDU,
    actionReturn-Request               [6] IMPLICIT ActionReturn-RequestPDU,
    informationReportList-Request       [7] IMPLICIT InformationReportList-RequestPDU,
    reject-Request                     [8] IMPLICIT Reject-RequestPDU,
    notificationRecovery-Request        [9] IMPLICIT NotificationRecovery-RequestPDU,
    eventNotificationList-Request      [10] IMPLICIT EventNotificationList-RequestPDU
}
```

4.1.3.9 Management ASE

4.1.3.9.1 Begin Set Attributes service

```
BeginSetAttributes-RequestPDU ::= Integer8 {
    loadingNonInteracting      (0),
    appendingInteracting       (1),
    freshLoadingInteracting     (2)
}
```

BeginSetAttributes -ResponsePDU ::= NULL

4.1.3.9.2 Create Service

4.1.3.9.2.1 Create Function Invocation PDU

```
CreateFunctionInvocation-RequestPDU ::= SEQUENCE {
    listOfRelatedObjects      [0] IMPLICIT SEQUENCE OF Gn_AccessSpecification,
    accessPrivilege           [1] IMPLICIT Fi_AccessPrivilege,
    optionalParametersMap     [10] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    fiName                    [2] IMPLICIT Gn_Name OPTIONAL,
    fiIndex                   [12] IMPLICIT Gn_NumericID OPTIONAL,
    userDescription           [3] IMPLICIT ANY OPTIONAL,
    reusable                  [4] IMPLICIT Gn_Reusable DEFAULT TRUE,
    deletable                 [5] IMPLICIT Gn_Deletable OPTIONAL,
    startServiceArgumentType  [6] IMPLICIT Gn_NumericID OPTIONAL,
    stopServiceArgumentType   [7] IMPLICIT Gn_NumericID OPTIONAL,
    resumeServiceArgumentType [8] IMPLICIT Gn_NumericID OPTIONAL,
    resetServiceArgumentType  [9] IMPLICIT Gn_NumericID OPTIONAL,
    killServiceArgumentType   [11] IMPLICIT Gn_NumericID OPTIONAL
}
```

CreateFunctionInvocation-ResponsePDU ::= Gn_NumericID

4.1.3.9.2.2 Define Variable List PDU

```
DefineVariableList-RequestPDU ::= SEQUENCE {
    listOfVariables      [0] IMPLICIT SEQUENCE OF Gn_AccessSpecification,
    accessPrivilege      [1] IMPLICIT Vr_AccessPrivilege,
    optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    variableListName      [2] IMPLICIT Gn_Name OPTIONAL,
    variableListIndex     [5] IMPLICIT Gn_NumericID OPTIONAL,
    userDescription       [3] IMPLICIT ANY OPTIONAL,
    deletable             [4] IMPLICIT Gn_Deletable OPTIONAL
}
```

DefineVariableList-ResponsePDU ::= Gn_NumericID

4.1.3.9.3 Delete Service

4.1.3.9.3.1 Delete Function Invocation PDU

DeleteFunctionInvocation-RequestPDU ::= Gn_AccessSpecification

DeleteFunctionInvocation-ResponsePDU ::= NULL

4.1.3.9.3.2 Delete Variable List PDU

DeleteVariableList-RequestPDU ::= Gn_AccessSpecification

DeleteVariableList-ResponsePDU ::= NULL

4.1.3.9.4 End Set Attributes Service

EndSetAttributes-RequestPDU ::= NULL

EndSetAttributes-ResponsePDU ::= NULL

4.1.3.9.5 Get Attributes List Service

```

GetAttributes1-RequestPDU ::= CHOICE {
    listOfObjectsAndAttributes          [0] IMPLICIT SEQUENCE OF SEQUENCE {
        falClass                       [6] IMPLICIT Gn_NumericID,
        keyAttribute                    Gn_KeyAttribute,
        listOfRequestedAttributes       [7] IMPLICIT Mn_AttributesMap
    },
    rangeOfObjectsAndAttributes        [1] IMPLICIT SEQUENCE {
        startingNumericId              [0] IMPLICIT Gn_NumericID,
        listOfRequestedAttributes      [1] IMPLICIT Mn_AttributesMap
    }
}

GetAttributes2-RequestPDU ::= SEQUENCE {
    listOfAttributes                    [0] IMPLICIT Mn_SelectedAttributes,
    accessSpecification CHOICE {
        numericID                      [1] IMPLICIT Gn_NumericID,
        variableName                   [2] IMPLICIT Gn_Name,
        variableListName               [3] IMPLICIT Gn_Name,
        domainName                     [4] IMPLICIT Gn_Name,
        fiName                          [5] IMPLICIT Gn_Name,
        eventName                      [6] IMPLICIT Gn_Name,
        startIndex                     [7] IMPLICIT Gn_NumericID
    }
}

GetAttributes3-RequestPDU ::= SEQUENCE {
    objectClass                        [2] IMPLICIT Gn_ObjectClass,
    start objectID                     Gn_AccessSpecification
}

GetAttributes1-ResponsePDU ::= SEQUENCE {
    listOfAttributeIdAndValues         [0] IMPLICIT Mn_AttributeValues,
    more                              [1] IMPLICIT Gn_MoreFollows
}

GetAttributes2-ResponsePDU ::= SEQUENCE {
    listOfObjectDefinition             [0] IMPLICIT SEQUENCE OF Gn_ObjectDefinition,
    more                              [1] IMPLICIT Gn_MoreFollows
}

GetAttributes3-ResponsePDU ::= CHOICE {
    listofobjectID                    [0] IMPLICIT SEQUENCE {
        listofID                       [2] IMPLICIT SEQUENCE OF Gn_AccessSpecification,
        morefollows                    [3] IMPLICIT Gn_MoreFollows
    }, --The object class value in the request is <> 0
    listofobjectDefinition            [1] IMPLICIT SEQUENCE {
        listofDefinition               [4] IMPLICIT SEQUENCE OF Mn_AttributesValues,
        morefollows                    [5] IMPLICIT Gn-MoreFollows
    } --The object classvalue in the request is = 0
}

```

4.1.3.9.6 Set Attributes Service

```

SetAttributes1-RequestPDU ::= SEQUENCE OF SEQUENCE {
    falClass                          [6] IMPLICIT Gn_NumericID,
    keyAttributes                     Gn_KeyAttributes,
    listOfAttributeUpdates            [7] IMPLICIT Mn_AttributeValues
}

SetAttributes2-RequestPDU ::= SEQUENCE OF {
    objectDefinition                  [0] IMPLICIT Gn_ObjectDefinition
}

```


SetAttributes1-ResponsePDU ::= NULL

SetAttributes2-ResponsePDU ::= NULL

4.1.3.10 Application Process ASE

4.1.3.10.1 Get Status Service

Status-RequestPDU ::= NULL

Status-ResponsePDU ::= SEQUENCE {
 logicalStatus
 state-changes-allowed [0] IMPLICIT Unsigned8 {
 (0),
 no-state-changes-allowed (1),
 limited-services-permitted (2),
 od-loading-non-interacting (4),
 od-loading-interacting (5)
 },
 physicalStatus [1] IMPLICIT Unsigned8 {
 operational (0),
 partially-operational (1),
 inoperable (2),
 needs-commissioning (3)
 },
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 localDetail [2] IMPLICIT BitString OPTIONAL --bit 1
 }

4.1.3.10.2 Identify Service

Identify-RequestPDU ::= NULL

Identify-ResponsePDU ::= SEQUENCE {
 vendorName [0] IMPLICIT VisibleString,
 modelIdentifier [1] IMPLICIT VisibleString,
 vendorRevision [2] IMPLICIT VisibleString,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 serialNumber [3] IMPLICIT OctetString OPTIONAL -- bit 1
 }

4.1.3.10.3 Initiate Service

Initiate-RequestPDU ::= SEQUENCE {
 versionObjectDefinitionsCalling [0] IMPLICIT Integer16,
 apDescriptorCalling [1] IMPLICIT OctetString,
 accessProtectionSupportedCalling [2] IMPLICIT Ap_AccessProtectionSupported,
 passwordAndAccessGroupsCalling [3] IMPLICIT Ap_AccessControl,
 configuredMaxPduSizeSending [4] IMPLICIT Unsigned8, -- See NOTE
 configuredMaxPduSizeReceiving [5] IMPLICIT Unsigned8, -- See NOTE
 listOfSupportedServicesCalling [6] IMPLICIT Mn_PduSupportedMap,
 configuredMaxOutstandingServicesRequesting [7] IMPLICIT Unsigned8,
 configuredMaxOutstandingServicesResponding [8] IMPLICIT Unsigned8,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 userDetail [11] IMPLICIT OctetString OPTIONAL, -- bit 1
 configuredNestingLevel [12] IMPLICIT Unsigned8 DEFAULT 1 -- bit 2
 }

Initiate-ResponsePDU ::= SEQUENCE {
 versionObjectDefinitionsCalled [0] IMPLICIT Integer16,
 profileNumberCalled [1] IMPLICIT OctetString,
 accessPrivilegeSupportedCalled [2] IMPLICIT Ap_AccessProtectionSupported,
 passwordAndAccessGroupsCalled [3] IMPLICIT Ap_AccessControl,
 negotiatedMaxOutstandingServicesRequesting [4] IMPLICIT Unsigned8,
 negotiatedMaxOutstandingServicesResponding [5] IMPLICIT Unsigned8,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 userDetail [11] IMPLICIT OctetString OPTIONAL, -- bit 1
 negotiatedNestingLevel [12] IMPLICIT Unsigned8 DEFAULT 1, -- bit 2
 negotiatedListOfServices [13] IMPLICIT Mn_PduSupportedMap OPTIONAL -- bit 3
 }


```

Initiate-ErrorPDU ::= SEQUENCE {
    errorCode
        other
        max-fal-pdu-size-insufficient
        service-not-supported
        version-obj-def-incompatible
        user-initiate-denied
        password-error
        profile-number-incompatible
    },
    maxPduLengthSendingCalled
    maxPduLengthReceivingCalled
    listOfSupportedServicesCalled
}

[0] IMPLICIT Integer8 {
    (0),
    (1),
    (2),
    (3),
    (4),
    (5),
    (6)
}

[1] IMPLICIT Unsigned8, -- See NOTE
[2] IMPLICIT Unsigned8, -- See NOTE
[3] IMPLICIT Mn_PduSupportedMap

```

NOTE For QUB-Seg the PDU length is given in multiples of 256 octets, i.e. value 4 means $4 \times 256 = 1024$ octets

4.1.3.10.4 Status Notification Service

UnsolicitedStatus-RequestPDU ::= Status-ResponsePDU

4.1.3.10.5 Subscribe Service

```

Subscribe-RequestPDU ::= CHOICE {
    joining
        publishedObject
        scheduleInterval
        optionalParametersMap
        maximumARs
        maximumPermissibleJitter
        desiredPublishingStartTime
        earliestPublishingStartTime
        latestPublishingStartTime
    },
    listOfLeavingAREPs
}

[0] IMPLICIT SEQUENCE {
    Gn_KeyAttributes,
    [11] IMPLICIT DI_CyclicInterval,
    [12] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    [13] IMPLICIT Unsigned8 DEFAULT 1 OPTIONAL, -- bit 1
    [14] IMPLICIT DI_PermissibleJitter OPTIONAL, -- bit 2
    [15] IMPLICIT FieldbusTime OPTIONAL, -- bit 3
    [16] IMPLICIT FieldbusTime OPTIONAL, -- bit 4
    [17] IMPLICIT FieldbusTime OPTIONAL -- bit 5
}

[1] IMPLICIT SEQUENCE OF Gn_NumericID

Subscribe-ResponsePDU ::= SEQUENCE {
    optionalParametersMap
    dataType
    dataLength
    dedicated
    listOfArInformation
        publishingAREPid
        publishingDIMapping
        scheduledStartTime
    } OPTIONAL
}

[0] Gn_OptionalParametersMap8 OPTIONAL,
[1] Gn_NumericID OPTIONAL, -- bit 1
[2] DI_SduSize OPTIONAL, -- bit 2
[3] Ar_Dedicated DEFAULT TRUE, -- bit 3
[4] IMPLICIT SEQUENCE OF SEQUENCE {
    [0] Gn_NumericID,
    [1] Gn_NumericID,
    [2] FieldbusTime
} -- bit 4

```

4.1.3.10.6 Reject Service

```

RejectPDU ::= SEQUENCE {
    original-invokedID
    reject-code
        other
        illegal-confirmed-service-request
        pdu-error
        pdu-size
    }
}

[0] IMPLICIT Integer8,
[1] IMPLICIT Integer8 {
    (0),
    (1),
    (2),
    (5)
}

```

4.1.3.10.7 Conclude Service

Conclude-Request ::= Null

Conclude-Response ::= Null

4.1.3.11 Load Region ASE

4.1.3.11.1 Discard Service

Discard-RequestPDU ::= Gn_KeyAttribute

-- Load Region LocalID

Discard-ResponsePDU ::= NULL

4.1.3.11.2 Initiate Load Service

4.1.3.11.2.1 InitiateClientPullDownloadStatic PDUs

InitiateClientPullDownloadStatic-RequestPDU ::= Gn_AccessSpecification

InitiateClientPullDownloadStatic-ResponsePDU ::= NULL

4.1.3.11.2.2 InitiateClientPullDownloadDynamic PDUs

```
InitiateClientPullDownloadDynamic-RequestPDU ::= SEQUENCE {
    loadRegionKeyAttribute      Gn_AccessSpecification,
    userData                    [0] OctetString,
    sharable                    [1] BOOLEAN,
    accessProtection             [2] Fi-AccessPrivilege
}
```

InitiateClientPullDownloadDynamic-ResponsePDU ::= NULL

4.1.3.11.2.3 InitiateClientPullUploadStatic PDUs

InitiateClientPullUploadStatic-RequestPDU ::= Gn_AccessSpecification

InitiateClientPullUploadStatic-ResponsePDU ::= NULL

4.1.3.11.2.4 InitiateClientPullUploadDynamic PDUs

InitiateClientPullUploadDynamic-RequestPDU ::= Gn_AccessSpecification

```
InitiateClientPullUploadDynamic-ResponsePDU ::= SEQUENCE {
    ulsmNumericID              [0] Gn_NumericID,
    userData                    [1] OctetString
}
```

4.1.3.11.2.5 InitiateServerPullDownload PDUs

```
InitiateServerPullDownload-RequestPDU ::= SEQUENCE {
    loadRegionKeyAttribute      Gn_AccessSpecification,
    optionalParametersMap       [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    additionalInformation        [11] IMPLICIT VisibleString OPTIONAL
}
```

-- bit 1

InitiateServerPullDownload-ResponsePDU ::= NULL

4.1.3.11.2.6 InitiateServerPullUpload PDUs

```
InitiateServerPullUpload-RequestPDU ::= SEQUENCE {
    loadRegionKeyAttribute      Gn_AccessSpecification,
    optionalParametersMap       [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    additionalInformation        [2] IMPLICIT VisibleString OPTIONAL
}
```

-- bit 1

InitiateServerPullUpload-ResponsePDU ::= NULL

4.1.3.11.2.7 InitiateClientPushDownload PDUs

InitiateClientPushDownload-RequestPDU ::= Gn_AccessSpecification

InitiateClientPushDownload-ResponsePDU ::= NULL

4.1.3.11.3 Pull Segment Service

4.1.3.11.3.1 PullDownloadSegment PDUs

```
PullDownloadSegment-RequestPDU ::= SEQUENCE {
    loadRegionKeyAttribute      Gn_AccessSpecification,
    optionalParametersMap       [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    segmentNumber               [11] IMPLICIT Fi_SegmentNumber OPTIONAL
}
```

-- bit 1

```
PullDownloadSegment-ResponsePDU ::= SEQUENCE {
    loadData                    [0] IMPLICIT OctetString,
    moreFollows                 [1] IMPLICIT Gn_MoreFollows
}
```

4.1.3.11.3.2 PullUploadSegment PDUs

PullUploadSegment-RequestPDU ::= Gn_AccessSpecification

```
PullUploadSegment-ResponsePDU ::= SEQUENCE {
    loadData                    [0] IMPLICIT OctetString,
    moreFollows                 [1] IMPLICIT Gn_MoreFollows
}
```

4.1.3.11.4 Push Segment Service**4.1.3.11.4.1 PushDownloadSegment PDUs**

PushDownloadSegment-RequestPDU ::= SEQUENCE {
 loadRegionKeyAttribute Gn_AccessSpecification,
 loadData [2] IMPLICIT OctetString,
 moreFollows [3] IMPLICIT Gn_MoreFollows,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 segmentNumber [11] IMPLICIT Fi_SegmentNumber OPTIONAL
 } -- bit 1

PushDownloadSegment-ResponsePDU ::= NULL

4.1.3.11.5 Terminate Load Service**4.1.3.11.5.1 TerminatePullDownload PDUs**

TerminatePullDownload-RequestPDU ::= SEQUENCE {
 loadRegionKeyAttribute Gn_AccessSpecification,
 terminateReason [2] IMPLICIT Lr_TerminateReason
 }

TerminatePullDownload-ResponsePDU ::= NULL

4.1.3.11.5.2 TerminatePushDownload PDUs

TerminatePushDownload-RequestPDU ::= Gn_AccessSpecification

TerminatePushDownload-ResponsePDU ::= Fi_TerminateReason

4.1.3.11.5.3 TerminatePullUpload PDUs

TerminatePullUpload-RequestPDU ::= Gn_AccessSpecification

TerminatePullUpload-ResponsePDU ::= NULL

4.1.3.12 Function Invocation ASE**4.1.3.12.1 ActionInvoke Service**

ActionInvoke-RequestPDU ::= SEQUENCE {
 keyAttribute Gn_KeyAttribute,
 actionInvocationID [6] IMPLICIT Fi_ActionInvokeID,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 executionArgument [7] IMPLICIT OctetString OPTIONAL
 } -- bit 1

4.1.3.12.2 ActionReturn Service

ActionReturn-RequestPDU ::= SEQUENCE {
 actionInvocationID [0] IMPLICIT Fi_ActionInvokeID,
 actionResults CHOICE {
 actionSuccess [1] IMPLICIT ANY,
 actionFailure [2] IMPLICIT ErrorType
 }
 }

4.1.3.12.3 Kill Service

Kill-RequestPDU ::= SEQUENCE {
 keyAttribute Gn_KeyAttribute,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 executionArgument [7] IMPLICIT OctetString OPTIONAL
 } -- bit 1

Kill-ResponsePDU ::= NULL

4.1.3.12.4 Reset Service

Reset-RequestPDU ::= SEQUENCE {
 keyAttribute Gn_KeyAttribute,
 optionalParametersMap [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
 executionArgument [7] IMPLICIT OctetString OPTIONAL
 } -- bit 1

Reset-ResponsePDU ::= NULL

4.1.3.12.5 Resume Service

```
Resume-RequestPDU ::= SEQUENCE {
    keyAttribute
    optionalParametersMap
    executionArgument
    Gn_KeyAttribute,
    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    [7] IMPLICIT OctetString OPTIONAL
}
-- bit 1
```

Resume-ResponsePDU ::= NULL

4.1.3.12.6 Start Service

```
Start-RequestPDU ::= SEQUENCE {
    keyAttribute
    optionalParametersMap
    executionArgument
    Gn_KeyAttribute,
    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    [7] IMPLICIT OctetString OPTIONAL
}
-- bit 1
```

Start-ResponsePDU ::= NULL

4.1.3.12.7 Stop Service

```
Stop-RequestPDU ::= SEQUENCE {
    keyAttribute
    optionalParametersMap
    executionArgument
    Gn_KeyAttribute,
    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    [7] IMPLICIT OctetString OPTIONAL
}
-- bit 1
```

Stop-ResponsePDU ::= NULL

4.1.3.13 Variable Access ASE

4.1.3.13.1 Exchange Service

```
Exchange-RequestPDU ::= SEQUENCE {
    specifierForVariableToWrite
    valueToWrite
    specifierForVariableToRead
    optionalParametersMap
    objectRevisionForVariableWriting
    objectRevisionRequested
    [0] Gn_KeyAttribute,
    [1] IMPLICIT ANY,
    [2] Gn_KeyAttribute,
    [2] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    [3] IMPLICIT Gn_ObjectRevision OPTIONAL,
    [4] IMPLICIT Gn_RequestFlag DEFAULT FALSE
}
-- bit 1
-- bit 2
```

```
Exchange-ResponsePDU ::= SEQUENCE {
    writeResult
    readResult CHOICE {
        status
        value
        valueWithRevision
        value
        objectRevision
    }
    [0] IMPLICIT Vr_WriteResult,
    [1] IMPLICIT Er_Access,
    [2] IMPLICIT ANY,
    [3] IMPLICIT SEQUENCE {
        [4] IMPLICIT ANY,
        [5] IMPLICIT Gn_ObjectRevision
    }
}
}
```

4.1.3.13.2 Exchange List Service

```
ExchangeList-RequestPDU ::= SEQUENCE {
    writeList-Request
    readList-Request
    [0] IMPLICIT WriteList-RequestPDU,
    [1] IMPLICIT ReadList-RequestPDU
}
```

```
ExchangeList-Response ::= SEQUENCE {
    writeList-Response
    readList-Response
    [0] IMPLICIT WriteList-ResponsePDU,
    [1] IMPLICIT ReadList-ResponsePDU
}
```

4.1.3.13.3 Information Report Service

```
InformationReport-RequestPDU ::= SEQUENCE {
    value
    optionalParametersMap
    variableSpecifier
    subIndex
    objectRevision
    [4] IMPLICIT ANY,
    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    Gn_KeyAttribute OPTIONAL,
    [3] IMPLICIT Gn_SubIndex OPTIONAL,
    [11] IMPLICIT Gn_ObjectRevision OPTIONAL
}
-- bit 1
-- bit 2
-- bit 3
```

4.1.3.13.4 Information Report with Type Service

```

InformationReportWithType-RequestPDU ::= SEQUENCE{
    TypeDescription          [4] Gn_TypeDescription,
    value                    [5] IMPLICIT ANY,
    optionalParametersMap    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    variableSpecifier        Gn_KeyAttribute OPTIONAL,           -- bit 1
    subIndex                 [3] IMPLICIT Gn_SubIndex OPTIONAL,  -- bit 2
    objectRevision           [11] IMPLICIT Gn_ObjectRevision OPTIONAL -- bit 3
}

```

4.1.3.13.5 Information List Report Service

```

InformationReportList-RequestPDU ::= SEQUENCE {
    ListOfVariableSpecifier CHOICE {
        variableList          [0] IMPLICIT Gn_KeyAttribute,
        listOfVariable        [1] IMPLICIT Gn_ListOfVariableAccess
    },
    listOfDataType            [7] IMPLICIT SEQUENCE OF Gn_FullyNestedTypeDescription,
    listOfData                [8] IMPLICIT SEQUENCE OF ANY,
    optionalParametersMap     [10] IMPLICIT Gn_OptionalParametersMap8,
    userDetails               [11] IMPLICIT OctetString OPTIONAL,   -- bit 1
    listOfObjectRevision      [12] SEQUENCE OF Gn_ObjectRevision OPTIONAL -- bit 2
}

```

-- The listOfDataType should either contain one element per variable or should be empty. The listOfObjectRevision should contain one element per variable or should be absent.

4.1.3.13.6 Read Service**4.1.3.13.6.1 Read PDUs**

```

Read1-RequestPDU ::= SEQUENCE {
    variableSpecifier        Gn_KeyAttribute,
    optionalParametersMap    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    objectRevisionRequested  [11] IMPLICIT Gn_RequestFlag DEFAULT FALSE OPTIONAL -- bit 1
}

Read2-RequestPDU ::= SEQUENCE {
    variableSpecifier        Gn_KeyAttribute,
    optionalParametersMap    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    subIndex                 [3] IMPLICIT Gn_SubIndex OPTIONAL                     -- bit 1
}

Read1-ResponsePDU ::= SEQUENCE {
    value                    [0] IMPLICIT ANY,
    optionalParametersMap    [1] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    objectRevision           [2] IMPLICIT Gn_ObjectRevision OPTIONAL               -- bit 1
}

```

Read2-ResponsePDU ::= ANY

4.1.3.13.6.2 PhysicalRead PDUs

```

PhysicalRead-RequestPDU ::= SEQUENCE {
    local-address            [0] IMPLICIT Gn_PhysicalAddress,
    length                   [1] IMPLICIT Gn_Length
}

```

PhysicalRead-ResponsePDU ::= ANY

4.1.3.13.6.3 ReadWithType PDUs

```

ReadWithType-RequestPDU ::= SEQUENCE{
    variableSpecifier        Gn_KeyAttribute,
    optionalParametersMap    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    subIndex                 [3] IMPLICIT Gn_SubIndex OPTIONAL                     -- bit 1
}

ReadWithType-ResponsePDU ::= SEQUENCE{
    typeDescription          [0] Gn_TypeDescription,
    value                    [1] IMPLICIT ANY,
    optionalParametersMap    [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    objectRevision           [11] IMPLICIT Gn_ObjectRevision OPTIONAL             -- bit 1
}

```

4.1.3.13.7 Read List Service

```

ReadList-RequestPDU ::= SEQUENCE {
    ListOfVariableSpecifier CHOICE {
        variableList
        listOfVariable
        listOfNumericAddress
    },
    additionalInfoRequested
    typeDescriptionRequested
    objectRevisionRequested
}

[0] IMPLICIT Gn_KeyAttributes,
[1] IMPLICIT Gn_ListOfVariableAccess,
[2] IMPLICIT SEQUENCE OF Gn_NumericAddress

BitString8 {
    (0),
    (1)
}

```

```

ReadList-ResponsePDU ::= SEQUENCE {
    ListOfAccessResult
    listOfDataType
    listOfData
    listOfObjectRevision
}

[0] IMPLICIT BitString,
[1] IMPLICIT SEQUENCE OF Gn_FullyNestedTypeDescription,
[2] IMPLICIT SEQUENCE OF ANY,
[3] IMPLICIT SEQUENCE OF Gn_ObjectRevision

```

- The listOfAccessResult parameter specifies for each requested variable if the access was successful (bit = 1) or not (bit = 0).
- The listOfDataType should either contain one element per variable or should be empty. The same applies for the listOfObjectRevision.
- The listOfData parameter specifies for each requested variable the value of the variable if the access succeeds or an ErrorClass if it fails. The ErrorClass is coded as an unsigned16 where the most significant byte contains the "error class" and the least significant byte contains the "error code" (see ErrorClass definition).

4.1.3.13.7.1 Write Service

4.1.3.13.7.2 Write PDUs

```

Write-RequestPDU ::= SEQUENCE {
    variableSpecifier
    value
    optionalParametersMap
    subIndex
    objectRevision
}

Gn_KeyAttribute,
[4] IMPLICIT ANY,
[10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
[3] IMPLICIT Gn_SubIndex OPTIONAL,
[11] IMPLICIT Gn_ObjectRevision OPTIONAL

-- bit 1
-- bit 2

```

Write-ResponsePDU ::= NULL

4.1.3.13.7.3 PhysicalWrite PDUs

```

PhysicalWrite-RequestPDU ::= SEQUENCE {
    local-address
    data
}

[0] IMPLICIT Gn_PhysicalAddress,
[1] IMPLICIT ANY

```

PhysicalWrite-ResponsePDU ::= NULL

4.1.3.13.7.4 WriteWithType PDUs

```

WriteWithType-RequestPDU ::= SEQUENCE{
    VariableSpecifier
    typeDescription
    value
    optionalParametersMap
    subIndex
    objectRevision
}

Gn_KeyAttribute,
[4] Gn_TypeDescription,
[5] IMPLICIT ANY,
[10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
[3] IMPLICIT Gn_SubIndex OPTIONAL,
[11] IMPLICIT Gn_ObjectRevision OPTIONAL

- bit 1
- bit 2

```

WriteWithType-ResponsePDU ::= NULL

4.1.3.13.8 Write List Service

```

WriteList-RequestPDU ::= SEQUENCE {
    ListOfVariableSpecifier CHOICE {
        variableList          [0] IMPLICIT Gn_KeyAttributes,
        listOfVariable        [1] IMPLICIT Gn_ListOfVariableAccess,
        listOfNumericAddress  [2] IMPLICIT SEQUENCE OF Gn_NumericAddress
    },
    listOfDataType            [3] IMPLICIT SEQUENCE OF Gn_FullyNestedTypeDescription,
    listOfValue               [4] IMPLICIT SEQUENCE OF ANY,
    listOfObjectRevision      [5] IMPLICIT SEQUENCE OF Gn_ObjectRevision
}

```

-- The listOfDataType should either contain one element per variable or should be empty. The same applies for the listOfObjectRevision.

```

WriteList-ResponsePDU ::= SEQUENCE {
    ListOfAccessResult        [0] IMPLICIT BitString,
    listOfAccessError         [1] IMPLICIT SEQUENCE OF ErrorClass
}

```

-- The listOfAccessResult parameter specifies for each requested variable if the access was successful (bit = 1) or not (bit = 0).

-- The listOfAccessError contains one error code for each failed access. If a variable access was successful, there is no corresponding error code in this list.

-- The ErrorClass is coded as an unsigned16 where the most significant byte contains the "error class" and the least significant byte contains the "error code" (see ErrorClass definition).

4.1.3.14 Event Management ASE

4.1.3.14.1 Confirmed Acknowledge Event service

```

AcknowledgeEventNotification-RequestPDU ::= SEQUENCE {
    keyAttribute              Gn_AccessSpecification,
    reportedEventCount        [2] IMPLICIT Ev_EventCount
}

```

AcknowledgeEventNotification-ResponsePDU ::= NULL

4.1.3.14.2 EnableEvent Service

```

EnableEvent-RequestPDU ::= SEQUENCE {
    keyAttribute              Gn_AccessSpecification,
    enabled                   [2] IMPLICIT Gn_Enabled
}

```

EnableEvent-ResponsePDU ::= NULL

4.1.3.14.3 EnableEventList Service

```

EnableEventList-RequestPDU ::= SEQUENCE {
    listOfKeyAttribute        [0] IMPLICIT SEQUENCE OF Gn_AccessSpecification,
    listOfEnabledFlags        [1] IMPLICIT BitString
}

```

```

EnableEventList-ResponsePDU ::= SEQUENCE {
    listOfEnableResult        [0] IMPLICIT BitString,
    listOfError               [1] IMPLICIT ErrorClass
}

```


4.1.3.14.4 Event Notification Service

```

EventNotificationList-RequestPDU ::= SEQUENCE {
    eventNotifierID          IMPLICIT Gn_NumericID,
    eventDataContained        Boolean,
    notificationTypeInfo CHOICE {
        basic                [0] NULL,
        sequenced            [1] Ev_SequenceNumber,
        time-of-notification [2] Ev_TimeTag,
        compound              [3] IMPLICIT SEQUENCE {
            sequence-number  Ev_SequenceNumber,
            time-tag         Ev_TimeTag
        }
    },
    listOfEventID            SEQUENCE OF Gn_AccessSpecification,
    listOfEv_EventDataType   SEQUENCE OF Gn_FullyNestedTypeDescription,
    listOfEventContents CHOICE {
        simple               [0] SEQUENCE OF ANY, -- Null, Ev_EventData,
        reportedEventCount   [1] SEQUENCE OF ANY, -- Ev_EventCount, Ev_EventData
        eventDetectionTime    [2] SEQUENCE OF ANY, -- Ev_TimeTag, Ev_EventData
        composite             [3] SEQUENCE OF ANY -- Ev_EventCount, Ev_TimeTag, Ev_EventData
    }
}

```

-- If the eventDataContained parameter is TRUE, the Ev_EventData value shall be present.

-- The components of event contents shall be appended together without any gap.

```

EventNotification-RequestPDU ::= SEQUENCE {
    KeyAttribute              Gn_KeyAttribute,
    eventNumber               [2] IMPLICIT Unsigned8,
    optionalParametersMap     [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    eventData                 [3] IMPLICIT ANY OPTIONAL, -- bit 1
    eventNotifierID           [11] IMPLICIT Gn_NumericID OPTIONAL, -- bit 2
    sequenceNumber            [12] IMPLICIT Ev_SequenceNumber OPTIONAL, -- bit 3
    notificationTime          [13] IMPLICIT Ev_TimeTag OPTIONAL -- bit 4
}

```

4.1.3.14.5 Event Notification with Type Service

```

EventNotificationWithType-RequestPDU ::= SEQUENCE {
    KeyAttribute              Gn_KeyAttribute,
    eventNumber               [2] IMPLICIT Unsigned8,
    typeDescription           [3] Gn_TypeDescription,
    optionalParametersMap     [10] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    value                     [4] IMPLICIT ANY OPTIONAL, -- bit 1
    eventNotifierID           [11] IMPLICIT Gn_NumericID OPTIONAL, -- bit 2
    sequenceNumber            [12] IMPLICIT Ev_SequenceNumber OPTIONAL, -- bit 3
    notificationTime          [13] IMPLICIT Ev_TimeTag OPTIONAL -- bit 4
}

```

4.1.3.14.6 Get Event Summary Service

```

GetEventSummary-RequestPDU ::= SEQUENCE {
    eventID                   [0] IMPLICIT Gn_NumericID,
    optionalParametersMap     [1] IMPLICIT OptionalParametersMap8 OPTIONAL,
    recoveredMessageRequested [2] IMPLICIT Gn_RequestFlag OPTIONAL, -- bit 1
    objectRevisionRequested   [3] IMPLICIT Gn_RequestFlag OPTIONAL -- bit 2
}

```

```

GetEventSummary-ResponsePDU ::= SEQUENCE {
    eventStatus               [0] IMPLICIT Ev_EventStatus,
    optionalParametersMap     [1] IMPLICIT OptionalParametersMap8 OPTIONAL,
    objectRevision            [2] IMPLICIT Gn_ObjectRevision OPTIONAL, -- bit 1
    recoveredMessage          [3] IMPLICIT Ev_EventMessage OPTIONAL -- bit 2
}

```

4.1.3.14.7 Get Event Summary List Service

```

GetEventSummaryList-RequestPDU ::= SEQUENCE {
    request events CHOICE {
        allEvents              [1] IMPLICIT Null,
        listOfEvents           [2] IMPLICIT SEQUENCE OF Gn_AccessSpecification
    }
}

```

```

    },
    additionalInfoRequest
        recoveredMessageRequested
        objectRevisionRequested
    }
}

```

```

GetEventSummaryList-ResponsePDU ::= CHOICE {
    listOfEvents
        listOfAccessResult
        listOfEventMessage
        listOfObjectRevision
    },
    allEvents
        listOfEventID
        listOfEventMessage
        listOfObjectRevision
    }
}

```

-- The listOfAccessResult parameter specifies for each requested event, if the access was successful (bit=1) or not (bit=0).

4.1.3.14.8 Notification Recovery Service

```

NotificationRecovery-RequestPDU ::= SEQUENCE {
    notifierID
        optionalParametersMap
        numberToRecover
        sequenceNumber
    }
    [0] IMPLICIT Gn_NumericID,
    [1] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    [2] IMPLICIT Ev_NumberToRecover DEFAULT 1 OPTIONAL,
    [3] IMPLICIT Ev_SequenceNumber OPTIONAL
    -- bit 1
    -- bit 2
}

```

4.1.3.14.9 Confirmed Acknowledge Event List Service

```

ConfirmedAcknowledgedEventList-RequestPDU ::= SEQUENCE {
    acknowledgmentPolicy
    listOfEventID
    listOfMessageSpecifier CHOICE {
        reportedEventCount
        eventDetectionTime
        eventCountAndDetectionTime
    },
    listOfAcknowledgmentData
}
    [0] IMPLICIT Ev_AcknowledgmentPolicy,
    [1] IMPLICIT SEQUENCE OF Gn_AccessSpecification,
    [2] IMPLICIT SEQUENCE OF Ev_EventCount,
    [3] IMPLICIT SEQUENCE OF Ev_TimeTag,
    [4] IMPLICIT SEQUENCE OF ANY -- Ev_EventCount, Ev_TimeTag
    [5] IMPLICIT SEQUENCE OF OctetString
}

```

```

ConfirmedAcknowledgedEventList-ResponsePDU ::= SEQUENCE {
    listOfAccessResult
    listOfError
}
    [0] IMPLICIT BitString,
    [1] IMPLICIT ErrorClass
}

```

4.1.3.14.10 Query Event Summary List Service

```

QueryEventSummaryList-RequestPDU ::= SEQUENCE {
    eventselectionCriterion
    requestedEvent
        allEvents
        listOfEvents
    },
    additionalInfoRequested
        recoveredMessageRequested
        objectRevisionRequested
    }
    [0] IMPLICIT BitString8,
    CHOICE {
        [1] IMPLICIT Null,
        [2] IMPLICIT SEQUENCE OF Gn_AccessSpecification
    },
    [3] IMPLICIT BitString8 {
        (0), --true for requested
        (1) --true for requested
    }
}

```

```

QueryEventSummaryList-ResponsePDU ::= CHOICE {
    listOfEvents
        listOfAccessResult
        listOfEventMessage
        listOfObjectRevision
    },
    allEvents
        listOfEventID
        listOfEventMessage
        listOfObjectRevision
    }
}

```

- The listofAccessResult parameter specifies for each event requested if the access was successful (bit=1) or not (bit=0).

4.1.3.15 Type Definitions

4.1.3.15.1 AP ASE Types

4.1.3.15.1.1 Ap_AccessProtectionSupported

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

4.1.3.15.1.2 Ap_AccessControl

- Ap_AccessControl ::= BitString {
- | | |
|-----------------|-------|
| password_Bit1 | (7), |
| password_Bit2 | (6), |
| password_Bit3 | (5), |
| password_Bit4 | (4), |
| password_Bit5 | (3), |
| password_Bit6 | (2), |
| password_Bit7 | (1), |
| password_Bit8 | (0), |
| access_Groups-1 | (15), |
| access_Groups-2 | (14), |
| access_Groups-3 | (13), |
| access_Groups-4 | (12), |
| access_Groups-5 | (11), |
| access_Groups-6 | (10), |
| access_Groups-7 | (9), |
| access_Groups-8 | (8) |
- }
- The Password (Unsigned8) is encoded as a bit string.

4.1.3.15.2 AR ASE Types

4.1.3.15.2.1 Ar_Dedicated

- Ar_Dedicated ::= Boolean
- The value of TRUE means the AREP is dedicated.
 - The value of FALSE means the AREP is not dedicated.

4.1.3.15.2.2 AREP

- RespondingAREP ::= Unsigned32
- Responding DLCEP address
- RequestingAREP ::= Unsigned32
- Requesting DLCEP address

4.1.3.15.2.3 Abort Types

Identifier

- Identifier ::= Unsigned8 {
- | | |
|------|------|
| user | (1), |
| nma | (5) |
- }

Reason Code

- ReasonCode ::= Unsigned8

Additional Detail

- AdditionalDetaill ::= OctetString

4.1.3.15.2.4 Ar_ServiceTimeOut

- Ar_ServiceTimeOut ::= Unsigned8

4.1.3.15.3 Data Link Layer Types

4.1.3.15.3.1 DI_CyclicInterval

- DI_CyclicInterval ::= Unsigned32
- Granularity is defined in IEC 61158-3 and IEC 61158-4.

4.1.3.15.3.2 DI_DlsapAddress

- DI_DlsapAddress ::= Unsigned32
- The format of this type is defined in IEC 61158-4.

4.1.3.15.3.3 DI_PermissibleJitter

- DI_PermissibleJitter ::= Unsigned16
- Granularity is defined in IEC 61158-3 and IEC 61158-4.

4.1.3.15.3.4 DI_SduSize

DI_SduSize ::= Unsigned16

4.1.3.15.4 Data Type ASE Types**4.1.3.15.4.1 Dt_ListOfFields**

```

Dt_ListOfFields ::= SEQUENCE {
    fieldDataType
    optionalParametersMap
    fieldName
}

```

[0] IMPLICIT Gn_NumericID,
[2] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
[0] IMPLICIT Gn_Name OPTIONAL

-- bit 1

4.1.3.15.5 Event ASE Types**4.1.3.15.5.1 Ev_AcknowledgmentDataSpecifier**

Ev_AcknowledgmentDataSpecifier ::= Ev_DataSpecifier

4.1.3.15.5.2 Ev_AcknowledgementPolicy

Ev_AcknowledgementPolicy ::= Boolean

-- True means all messages are to be acknowledged.
-- False means only the specified message is to be acknowledged.

4.1.3.15.5.3 Ev_ContainedEventData

Ev_ContainedEventData ::= Boolean

-- True means the event message contains event data.
-- False means the event message does not contain event data.

4.1.3.15.5.4 Ev_ContainedMessageType

```

Ev_ContainedMessageType ::= Unsigned8 {
    simple
    reportedEventCount
    eventDetectionTime
    composite
    any
}

```

(1),
(2),
(3),
(4),
(5)

4.1.3.15.5.5 Ev_DataSpecifier

```

Ev_DataSpecifier ::= CHOICE {
    variableID
    dataTypeId
    dataLength
}

```

[0] IMPLICIT Gn_NumericID,
[1] IMPLICIT Gn_NumericID,
[2] IMPLICIT Unsigned8

4.1.3.15.5.6 Ev_EventCount

En_EventCount ::= Unsigned8

4.1.3.15.5.7 Ev_EventData

Ev_EventData ::= ANY

-- Ev_EventData contains the data associated with the respective event. If the data is not to be included in the notification, its length is 0.

4.1.3.15.5.8 Ev_EventMessage

```

Ev_EventMessage ::= CHOICE {
    eventKeyAttributeOnly
    eventKeyAttributeWithCount
    eventKeyAttributeWithTime
    eventKeyAttributeWithData
}

```

[0] IMPLICIT Gn_KeyAttribute,
[1] IMPLICIT Ev_KeyAttributeWithMessageCount,
[2] IMPLICIT Ev_KeyAttributeWithData,
[3] IMPLICIT Ev_KeyAttributeWithData

```

Ev_ListOfEventResponse ::= CHOICE {
    simple
    recoveredCount
    recoveredDetectionTime
    RecoveredDetectionTime
    composite
}

```

[1] IMPLICIT SEQUENCE OF Ev_Status,
[2] IMPLICIT SEQUENCE OF ANY, --Ev_Status, RecoveredCount
[3] IMPLICIT SEQUENCE OF ANY, --Ev_tStatus,
[4] IMPLICIT SEQUENCE OF ANY
--Ev_Status, RecoveredCount,

-- The components of Ev_Status & RecoveredCount shall be appended together without any gap.

- The components of Ev_Status & RecoveredDetectionTime shall be appended together without any gap.
- The components of Ev_Status & RecoveredCount & RecoveredDetectionTime shall be appended together without any gap.

```
Ev_Status ::= BitString8 {
    active                (0), --true for active
    detected              (1), --true for detected
    enabled               (2), --true for enabled
    acknowledged         (3) --true for acked
}
```

RecoveredCount ::= Unsigned8

RecoveredDetectionTime ::= Ev_TimeTag

4.1.3.15.5.9 Ev_EventStatus

```
Ev_EventStatus ::= SEQUENCE {
    active                [0] IMPLICIT Boolean, -- True means active
    detected              [1] IMPLICIT Boolean, -- True means detected
    enabled               [2] IMPLICIT Boolean, -- True means enabled
    acknowledgmentSupported [3] IMPLICIT Boolean, -- True means supported
    acknowledged         [4] IMPLICIT Boolean -- True means acknowledged
}
```

4.1.3.15.5.10 Ev_ErrorStatus

```
Ev_ErrorStatus ::= Boolean
-- True means the underlying condition is active.
-- False means the underlying condition is not active.
```

4.1.3.15.5.11 Ev_LastReportedEventInfo

```
Ev_LastReportedEventInfo ::= SEQUENCE {
    optionalParametersMap [0] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    count                 [1] IMPLICIT Unsigned8 OPTIONAL, -- bit 1
    detectionTime         [2] IMPLICIT Ev_TimeTag OPTIONAL -- bit 2
}
```

4.1.3.15.5.12 Ev_MessageType

```
Ev_MessageType ::= Unsigned8 {
    simpleMessage          (1),
    reportedEventCount      (2),
    eventDetectionTime      (3),
    composite               (4),
    simpleMessageWithData  (5),
    reportedEventCountWithData (6),
    eventDetectionTimeWithData (7),
    compositeWithData       (8)
}
```

4.1.3.15.5.13 Ev_NotificationType

```
Ev_NotificationType ::= Unsigned8 {
    basic                  (1),
    sequenced              (2),
    timeOfNotification      (3),
    compound               (4)
}
```

4.1.3.15.5.14 Ev_NumberToRecover

Ev_NumberToRecover ::= Unsigned8

4.1.3.15.5.15 Ev_SequenceNumber

Ev_SequenceNumber ::= Unsigned8

4.1.3.15.5.16 Ev_SelectionCriterion

```
Ev_SelectionCriterion ::= SEQUENCE {
    enabled                [0] IMPLICIT Boolean,
    detected               [1] IMPLICIT Boolean,
    acked                  [2] IMPLICIT Boolean,
    active                 [3] IMPLICIT Boolean
}
```

4.1.3.15.5.17 Ev_TimeTag

Ev_TimeTag ::= Unsigned16

4.1.3.15.6 Function Invocation ASE Types**4.1.3.15.6.1 Fi_ActionInvokeID**

Fi_ActionInvokeID ::= Unsigned8

4.1.3.15.6.2 Fi_AccessPrivilege

```

Fi_AccessPrivilege ::= BitString {
    rightToStartPassword          (23),
    rightToStopPassword           (22),
    rightToDeletePassword         (21),
    rightToStartAccessGroup       (19),
    rightToStopAccessGroup        (18),
    rightToDeleteAccessGroup      (17),
    rightToStartAllPartner        (31),
    rightToStopAllPartner         (30),
    rightToDeleteAllPartner       (29),
    password_Bit1                 (7),
    password_Bit2                 (6),
    password_Bit3                 (5),
    password_Bit4                 (4),
    password_Bit5                 (3),
    password_Bit6                 (2),
    password_Bit7                 (1),
    password_Bit8                 (0),
    access_Groups-1               (15),
    access_Groups-2               (14),
    access_Groups-3               (13),
    access_Groups-4               (12),
    access_Groups-5               (11),
    access_Groups-6               (10),
    access_Groups-7               (9),
    access_Groups-8               (8)
}

```

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

4.1.3.15.6.3 Fi_SegmentNumber

Fi_SegmentNumber ::= Unsigned32 -- The first segment number shall be one(1).

4.1.3.15.6.4 Fi_State

```

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

```

4.1.3.15.7 General Types**4.1.3.15.7.1 Gn_AccessSpecification**

```

Gn_AccessSpecification ::= CHOICE {
    index                        [0] IMPLICIT Index,
    name                        [1] IMPLICIT Name
}

```

4.1.3.15.7.2 Gn_Deletable

Gn_Deletable ::= Boolean

-- True means deletable.
-- False means not deletable.

4.1.3.15.7.3 Gn_Enabled

Gn_Enabled ::= Boolean

-- True means enabled.
-- False means disabled.

4.1.3.15.7.4 Gn_FullyNestedTypeDescription

```

Gn_FullyNestedTypeDescription ::= CHOICE {
    Boolean [1] Gn_Length,
    integer8 [2] Gn_Length,
    integer16 [3] Gn_Length,
    integer32 [4] Gn_Length,
    unsigned8 [5] Gn_Length,
    unsigned16 [6] Gn_Length,
    unsigned32 [7] Gn_Length,
    float32 [8] Gn_Length,
    float64 [15] Gn_Length,
    binaryDate [11] Gn_Length,
    timeOfDay [12] Gn_Length,
    timeDifference [13] Gn_Length,
    universalTime [16] Gn_Length,
    fieldbusTime [17] Gn_Length,
    time [21] Gn_Length,
    bitstring8 [22] Gn_Length,
    bitstring16 [23] Gn_Length,
    bitstring32 [24] Gn_Length,
    visiblestring1 [25] Gn_Length,
    visiblestring2 [26] Gn_Length,
    visiblestring4 [27] Gn_Length,
    visiblestring8 [28] Gn_Length,
    visiblestring16 [29] Gn_Length,
    octetstring1 [30] Gn_Length,
    octetstring2 [31] Gn_Length,
    octetstring4 [32] Gn_Length,
    octetstring8 [33] Gn_Length,
    octetstring16 [34] Gn_Length,
    bcd [35] Gn_Length,
    iso10646char [36] Gn_Length,
    binarytime0 [40] Gn_Length,
    binarytime1 [41] Gn_Length,
    binarytime2 [42] Gn_Length,
    binarytime3 [43] Gn_Length,
    binarytime4 [44] Gn_Length,
    binarytime5 [45] Gn_Length,
    binarytime6 [46] Gn_Length,
    binarytime7 [47] Gn_Length,
    binarytime8 [48] Gn_Length,
    binarytime9 [49] Gn_Length,
    visiblestring [9] Gn_Length,
    octetstring [10] Gn_Length,
    bitstring [14] Gn_Length,
    compactBooleanArray [37] Gn_Length,
    compactBCDArray [38] Gn_Length,
    iso646string [39] Gn_Length,
    array [55] IMPLICIT SEQUENCE {
        number-of-elements [0] IMPLICIT Unsigned8,
        element-type [1] Gn_FullyNestedTypeDescription
    },
    structure [56] IMPLICIT SEQUENCE OF Gn_FullyNestedTypeDescription
}

```

4.1.3.15.7.5 Gn_KeyAttribute

```

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

```

4.1.3.15.7.6 Gn_Length

```

Gn_Length ::= Unsigned8

```


4.1.3.15.7.7 Gn_ListOfVariableAccess

```

Gn_ListOfVariableAccess ::= SEQUENCE OF CHOICE {
    numericID          [0] IMPLICIT Gn_NumericID,
    name               [1] IMPLICIT Gn_Name,
    listName           [2] IMPLICIT Gn_Name,
    symbolicAddress    [5] IMPLICIT Gn_SymbolicAddress,
    partialNumericID   [6] IMPLICIT SEQUENCE {
        numericID      [0] IMPLICIT Gn_NumericID,
        partialAccess
    },
    partialName        [7] IMPLICIT SEQUENCE {
        name           [0] IMPLICIT Gn_Name,
        partialAccess
    },
    partialListName    [8] IMPLICIT SEQUENCE {
        listName       [0] IMPLICIT Gn_Name,
        partialAccess
    }
}

```

4.1.3.15.7.8 Gn_MoreFollows

Gn_MoreFollows ::= Boolean

4.1.3.15.7.9 Gn_Name

Gn_Name ::= OctetString

4.1.3.15.7.10 Gn_NameWithComponent

```

Gn_NameWithComponent ::= SEQUENCE {
    name               [0] IMPLICIT Gn_Name,
    component          [1] IMPLICIT Unsigned16
}

```

4.1.3.15.7.11 Gn_NumericAddress

```

Gn_NumericAddress ::= SEQUENCE {
    startAddress       [0] IMPLICIT Unsigned32, -- physical address of the starting location
    length             [1] IMPLICIT Unsigned16 -- octet length of a memory block
}

```

4.1.3.15.7.12 Gn_NumericID

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

4.1.3.15.7.13 Gn_NumericIDWithComponent

```

Gn_NumericIDWithComponent ::= SEQUENCE {
    numericID          [0] IMPLICIT Gn_NumericID,
    component          [1] IMPLICIT Unsigned16
}

```

4.1.3.15.7.14 Gn_ObjectDefinition

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

4.1.3.15.7.15 Gn_ObjectRevision

Gn_ObjectRevision ::= Unsigned8

- bits 8 - 5 are used as the Major Revision
- bits 4 - 1 are used as the Minor Revision
- each may have a value between zero and 15 inclusive

4.1.3.15.7.16 Gn_OptionalParametersMaps

Gn_OptionalParametersMap8 ::= BitString8

Gn_OptionalParametersMap16 ::= BitString16

Gn_OptionalParametersMap32 ::= BitString32

- The above types have special semantics to represent which optional parameters are included
- in the transfer syntax. Bit 1 corresponds to the first optional parameter of a given primitive,
- bit 2 corresponds to the second optional parameter, up to bit "n" corresponds to the "n-th"
- optional parameters. The value of one indicates that the corresponding optional parameter is
- present in the transfer syntax, and the value of zero indicates that it is not present.

4.1.3.15.7.17 Gn_PartialAccess

```
Gn_PartialAccess ::= CHOICE {
    componentID
    componentList
    nestingLevel
    accessedElements
}
```

[3] IMPLICIT Gn_SubIndex,
[20] IMPLICIT SEQUENCE {
[1] Unsigned8,
[2] SEQUENCE OF Gn_Sub-Index

-- The componentID of the partial access selects a single element on the first level of nesting for access. The componentList specifies one possibly complex base element on a specified level of nesting and, if necessary, selects a number of subelements from this base element. The first elements (their number is given by nestingLevel) of the accessedElements list define the path to the base element. The rest of the list selects a number of subelements to be accessed. If the rest of the list is empty, the complete base element is accessed.

4.1.3.15.7.18 Gn_PhysicalAddress

Gn_PhysicalAddress ::= Unsigned32

4.1.3.15.7.19 Gn_RequestFlag

Gn_RequestFlag ::= Boolean

-- TRUE means the specified parameter is requested in the response.
-- FALSE means the specified parameter is not requested in the response.

4.1.3.15.7.20 Gn_Reusable

Gn_Reusable ::= Boolean

-- True means reusable
-- False means not reusable.

4.1.3.15.7.21 Gn_SubIndex

Gn_SubIndex ::= Unsigned8

4.1.3.15.7.22 Gn_SymbolicAddress

Gn_SymbolicAddress ::= VisibleString

4.1.3.15.7.23 Gn_TypeDescription

Gn_TypeDescription ::= SEQUENCE OF CHOICE {

simple [1] IMPLICIT ANY,

-- Octet1: High byte Data Type Gn_NumericID
-- Octet2: Low byte Data Type Gn_NumericID
-- Octet3: Length

array [2] IMPLICIT ANY,

-- Octet1: High byte Data Type Gn_NumericID
-- Octet2: Low byte Data Type Gn_NumericID
-- Octet3: Length
-- Octet4: Number of Elements

structure [3] IMPLICIT ANY

-- Octet 1 : High byte Data Type Index (Element 1)
-- Octet 2 : Low byte Data Type Index (Element 1)
-- Octet 3 : Length (Element 1)
-- Octet 3n-2 : High byte Data Type Index (Element n)
-- Octet 3n-1 : Low byte Data Type Index (Element n)
-- Octet 3n-0 : Length (Element n)

}

4.1.3.15.7.24 Gn_ObjectClass

```
Gn_ObjectClass ::= Unsigned8 {
    anyObject                (0),
    loadRegion               (2),
    functionInvocation        (3),
    event                    (4),
    fixedLength&StringDataType (5),
    structuredDataType        (6),
    fixedLength&StringVariable (7),
    arrayVariable             (8),
    dataStructureVariable     (9),
    variableList              (10),
    arrayDataType             (12),
    eventList                 (17),
    notifier                  (18),
    action                    (19)
}
```

4.1.3.15.8 Load Region ASE Types**4.1.3.15.8.1 Lr_FixedLengthSegment**

```
Lr_FixedLengthSegment ::= Boolean
-- True means LR supports fixed-length segments only.
-- False means LR supports variable-length segments.
```

4.1.3.15.8.2 Lr_LocalAddress

```
Lr_LocalAddress ::= Unsigned32
```

4.1.3.15.8.3 Lr_Size

```
Lr_Size ::= Unsigned32
```

4.1.3.15.8.4 Lr_State

```
Lr_State ::= Unsigned8 {
    NotDownLoadable         (1),
    downLoadable             (2),
    downLoading              (3),
    downloadFailure          (4),
    downloadSuccess          (5),
    loaded                   (6),
    inUse                     (7)
}
```

4.1.3.15.8.5 Lr_TerminateReason

```
Lr_TerminateReason ::= Boolean
-- True means the load process was successfully completed.
-- False means the load process was completed with failure.
```

4.1.3.15.8.6 Lr_UploadState

```
Lr_UploadState ::= Unsigned8 {
    notUploadable            (1),
    uploadable                (2),
    uploading                 (3),
    completingUpload         (4)
}
```

4.1.3.15.9 Management ASE Types**4.1.3.15.9.1 Attribute Maps****Mn_ActionAttrMap**

```
Mn_ActionAttrMap ::= Bitstring16 {
    commonAttrValues         (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    reusable                  (4),
    listOfRelatedObjects      (5),
    actionInvokeServiceArgumentDataType (6),
    actionReturnServiceArgumentDataType (7)
}
```

Mn_ApAttrMap

```
Mn_ApAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfSupportedServices   (3),
    manufacturerIdentifier    (4),
    ApdescriptorReference     (5),
    networkAccessState        (6),
    listOfArEndpointInfo     (7),
    listOfInuseArEndpointInfo (8),
    listOfSupportedApoClassAndObjects (9),
    listOfDisapAddresses      (10),
    listOfEventSummarySelectionCriteria (11)
}
```

Mn_AttributeMap

```
Mn_AttributeMap ::= SEQUENCE {
    Mn_CommonAttrMap,
    attributeMap CHOICE {
        apAttrMap          [1] IMPLICIT Mn_ApAttrMap,
        dataTypeAttrMap    [2] IMPLICIT Mn_DataTypeAttrMap,
        variableAttrMap     [3] IMPLICIT Mn_VariableAttrMap,
        variableListAttrMap [4] IMPLICIT Mn_VariableListAttrMap,
        eventAttrMap        [5] IMPLICIT Mn_EventAttrMap,
        eventListAttrMap    [6] IMPLICIT Mn_EventListAttrMap,
        notifierAttrMap     [7] IMPLICIT Mn_NotifierAttrMap,
        loadRegionAttrMap   [8] IMPLICIT Mn_LoadRegionAttrMap,
        functionInvocationAttrMap [9] IMPLICIT Mn_FunctionInvocationAttrMap,
        actionAttrMap       [10] IMPLICIT Mn_ActionAttrMap
    }
}
```

Mn_CommonAttrMap

```
Mn_CommonAttrMap ::= BitString8 {
    numericID      (1),
    name           (2),
    instanceOf     (3),
    parentClass    (4),
    userDescription (5),
    objectRevision (6)
}
```

Mn_DataTypeAttrMap

```
Mn_DataTypeAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    dataTypeNumericId         (3),
    dataTypeName              (4),
    format                    (5),
    octetLength               (6),
    numberOfFields            (7),
    listOfFields              (8),
    numberOfArrayElements     (9),
    arrayElementDataType      (10)
}
```

Mn_EventAttrMap

```
Mn_EventAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    eventStatus               (4),
    messageType               (5),
    accessPrivilege           (6),
    lastReportedEventInfo     (7),
    eventDataSpecifier        (8),
    acknowledgmentDataSpecifier (9)
}
```

Mn_EventListAttrMap

```

Mn_EventListAttrMap ::= BitString8 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    numberOfEntries           (4),
    listOfEvents              (5)
}

```

Mn_FunctionInvocationAttrMap

```

Mn_FunctionInvocationAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    accessPrivilege           (4),
    deletable                 (5),
    reusable                  (6),
    functionInvocationState    (7),
    numberOfRelatedObjectsInUse (8),
    listOfRelatedObjects      (9),
    startArgumentDataTypes    (10),
    stopServiceArgumentDataTypes (11),
    resumeServiceArgumentDataTypes (12),
    resetServiceArgumentDataTypes (13),
    killServiceArgumentDataTypes (14),
    executionArgument          (15)
}

```

Mn_LoadRegionAttrMap

```

Mn_LoadRegionAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    loadRegionSize            (4),
    accessPrivilege           (5),
    localAddress              (6),
    loadRegionState           (7),
    uploadState               (8),
    numberOfRelatedObjectsInUse (9),
    listOfRelatedObjects      (10),
    contentsSize              (11),
    loadImageName             (12),
    fixedLengthSegment        (13),
    fixedSegmentLength        (14),
    intersegmentRequestTimeout (15),
    loadRegionSharable        (16)
}

```

Mn_NotifierAttrMap

```

Mn_NotifierAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    enabled                   (4),
    notificationArepClass     (5),
    notificationArep          (6),
    notificationType          (7),
    containedMessageType      (8),
    containedEventData        (9),
    lastNotificationSequenceNumber (10),
    listOfEvents              (11)
}

```

Mn_VariableAttrMap

```
Mn_VariableAttrMap ::= BitString16 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    symbolicAddress           (4),
    dataType                  (5),
    length                    (6),
    variableLengthConveyance  (7),
    numberOfArrayElements     (8),
    accessPrivilege           (9),
    localDetail               (10),
    fullyNestedTypeDescription (11)
}
```

Mn_VariableListAttrMap

```
Mn_VariableListAttrMap ::= BitString8 {
    commonAttrValues          (1),
    listOfSupportedAttributes (2),
    listOfOperationalServices (3),
    numberOfEntries           (4),
    listOfVariables           (5),
    deletable                 (6),
    accessPrivilege           (7)
}
```

4.1.3.15.9.2 Attribute Values

Mn_ActionAttrValues

```
Mn_ActionAttrValues ::= SEQUENCE {
    optionalParametersMap [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues      [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,
    listOfSupportedAttributes [2] IMPLICIT Mn_ActionAttrMap OPTIONAL,
    listOfOperationalServices [3] IMPLICIT Mn_ActionServiceMap OPTIONAL,
    reusable               [4] IMPLICIT Gn_Reusable DEFAULT TRUE,
    listOfRelatedObjects   [5] IMPLICIT SEQUENCE OF Gn_KeyAttribute OPTIONAL,
    actionInvokeServiceArgumentDataType [6] IMPLICIT SEQUENCE OF Gn_NumericID OPTIONAL,
    actionActionServiceArgumentDataType [7] IMPLICIT SEQUENCE OF Gn_NumericID OPTIONAL
}
```

Mn_ApAttrValues

```
Mn_ApAttrValues ::= SEQUENCE {
    optionalParametersMap [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues      [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,
    listOfSupportedAttributes [2] IMPLICIT Mn_ApAttrMap OPTIONAL,
    listOfSupportedServices [3] IMPLICIT Mn_ApServiceMap OPTIONAL,
    manufacturerIdentifier [4] IMPLICIT Mn_ManufacturerIdentifier OPTIONAL,
    ApdescriptorReference [5] IMPLICIT OctetString OPTIONAL,
    networkAccessState [6] IMPLICIT Mn_NetworkAccessState OPTIONAL,
    listOfArEndpointInfo [7] IMPLICIT SEQUENCE OF Mn_ArEndpointInfo OPTIONAL,
    listOfInuseArEndpointInfo [8] IMPLICIT SEQUENCE OF Mn_InuseArEndpointInfo OPTIONAL,
    listOfSupportedApoClassAndObjects [9] IMPLICIT SEQUENCE OF Mn_ApoClassAndObjects OPTIONAL,
    listOfDisapAddresses [10] IMPLICIT SEQUENCE OF DI_DisapAddress OPTIONAL,
    listOfEventSummarySelectionCriteria [11] IMPLICIT SEQUENCE OF Ev_SelectionCriterion OPTIONAL
}
```

Mn_AttributeValues

```
Mn_AttributeValues ::= CHOICE {
    ApAttrValues [1] IMPLICIT Mn_ApAttrValues,
    dataTypeAttrValues [2] IMPLICIT Mn_DataTypeAttrValues,
    variableAttrValues [3] IMPLICIT Mn_VariableAttrValues,
    variableListAttrValues [4] IMPLICIT Mn_VariableListAttrValues,
    eventAttrValues [5] IMPLICIT Mn_EventAttrValues,
    eventListAttrValues [6] IMPLICIT Mn_EventListAttrValues,
    notifierAttrValues [7] IMPLICIT Mn_NotifierAttrValues,
    loadRegionAttrValues [8] IMPLICIT Mn_LoadRegionAttrValues,
    functionInvocationAttrValues [9] IMPLICIT Mn_FunctionInvocationAttrValues,
    actionAttrValues [10] IMPLICIT Mn_ActionAttrValues
}
```

Mn_CommonAttrValues

```

Mn_CommonAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    numericID                  [1] IMPLICIT Gn_NumericID,                      -- bit 1
    name                       [2] IMPLICIT Gn_Name OPTIONAL,                  -- bit 2
    instanceOf                  [3] IMPLICIT Gn_NumericID OPTIONAL,            -- bit 3
    parentClass                  [4] IMPLICIT Gn_NumericID OPTIONAL,            -- bit 4
    userDescription              [5] IMPLICIT OctetString OPTIONAL,              -- bit 5
    objectRevision              [6] IMPLICIT Gn_ObjectRevision OPTIONAL          -- bit 6
}

```

Mn_DataTypeAttrValues

```

Mn_DataTypeAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,      -- bit 1
    listOfSupportedAttributes   [2] IMPLICIT Mn_DataTypeAttrMap OPTIONAL,       -- bit 2
    dataTypeNumericId          [3] IMPLICIT Gn_NumericID OPTIONAL,              -- bit 3
    dataTypeName                [4] IMPLICIT Gn_Name OPTIONAL,                  -- bit 4
    format                      [5] IMPLICIT Gn_NumericID OPTIONAL,              -- bit 5
    octetLength                 [6] IMPLICIT Unsigned8 OPTIONAL,                -- bit 6
    numberOfFields              [7] IMPLICIT Unsigned8 OPTIONAL,                -- bit 7
    listOfFields                [8] IMPLICIT SEQUENCE OF Dt_ListOfFields OPTIONAL -- bit 8
    numberOfArrayElements       [9] IMPLICIT Gn_NumericID OPTIONAL,              -- bit 9
    arrayElementType            [10] IMPLICIT Gn_NumericID OPTIONAL              -- bit 10
}

```

Mn_EventAttrValues

```

Mn_EventAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,      -- bit 1
    listOfSupportedAttributes   [2] IMPLICIT Mn_EventAttrMap OPTIONAL,          -- bit 2
    listOfOperationalServices   [3] IMPLICIT Mn_EventServiceMap OPTIONAL,       -- bit 3
    eventStatus                 [4] IMPLICIT Ev_EventStatus OPTIONAL,            -- bit 4
    messageType                 [5] IMPLICIT Ev_MessageType OPTIONAL,           -- bit 5
    accessPrivilege              [6] IMPLICIT Vr_VariableListAccessPrivilege OPTIONAL, -- bit 6
    lastReportedEventInfo       [7] IMPLICIT Ev_LastReportedEventInfo OPTIONAL, -- bit 7
    eventDataSpecifier           [8] IMPLICIT Ev_DataSpecifier OPTIONAL,          -- bit 8
    acknowledgmentDataSpecifier [9] IMPLICIT EV_AcknowledgementDataSpecifier OPTIONAL -- bit 9
}

```

Mn_EventListAttrValues

```

Mn_EventListAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap8 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,      -- bit 1
    listOfSupportedAttributes   [2] IMPLICIT Mn_EventListAttrMap OPTIONAL,      -- bit 2
    listOfOperationalServices   [3] IMPLICIT Mn_EventListServiceMap OPTIONAL,   -- bit 3
    numberOfEntries             [4] IMPLICIT Unsigned16 OPTIONAL,                -- bit 4
    listOfEvents                [5] IMPLICIT SEQUENCE OF Gn_NumericID OPTIONAL  -- bit 5
}

```

Mn_FunctionInvocationAttrValues

```

Mn_FunctionInvocationAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,      -- bit 1
    listOfSupportedAttributes   [2] IMPLICIT Mn_FunctionInvocationAttrMap OPTIONAL, -- bit 2
    listOfOperationalServices   [3] IMPLICIT Mn_FunctionInvocationServiceMap OPTIONAL, -- bit 3
    accessPrivilege              [4] IMPLICIT Vr_AccessPrivilege OPTIONAL,       -- bit 4
    deletable                   [5] IMPLICIT Mn_Deletable OPTIONAL,              -- bit 5
    reusable                     [6] IMPLICIT Mn_Reusable DEFAULT TRUE OPTIONAL,   -- bit 6
    functionInvocationState      [7] IMPLICIT Fi_State OPTIONAL,                 -- bit 7
    numberOfRelatedObjectsInUse [8] IMPLICIT Unsigned8 OPTIONAL,                 -- bit 8
    listOfRelatedObjects         [9] IMPLICIT SEQUENCE OF Gn_KeyAttribute OPTIONAL, -- bit 9
    startArgumentDataTypes       [10] IMPLICIT Gn_NumericID OPTIONAL,            -- bit 10
    stopServiceArgumentDataTypes [11] IMPLICIT Gn_NumericID OPTIONAL,            -- bit 11
    resumeServiceArgumentDataTypes [12] IMPLICIT Gn_NumericID OPTIONAL,          -- bit 12
    resetServiceArgumentDataTypes [13] IMPLICIT Gn_NumericID OPTIONAL,           -- bit 13
    killServiceArgumentDataTypes [14] IMPLICIT Gn_NumericID OPTIONAL,            -- bit 14
    executionArgument            [15] IMPLICIT OctetString OPTIONAL               -- bit 15
}

```


Mn_LoadRegionAttrValues

```

Mn_LoadRegionAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,           -- bit 1
    listOfSupportedAttributes  [2] IMPLICIT Mn_LoadRegionAttrValues OPTIONAL,       -- bit 2
    listOfOperationalServices  [3] IMPLICIT Mn_LoadRegionServiceMap OPTIONAL,       -- bit 3
    loadRegionSize             [4] IMPLICIT Lr_Size OPTIONAL,                      -- bit 4
    accessPrivilege            [5] IMPLICIT Vr_AccessPrivilege OPTIONAL,            -- bit 5
    localAddress               [6] IMPLICIT Lr_LocalAddress OPTIONAL,               -- bit 6
    loadRegionState            [7] IMPLICIT Lr_State OPTIONAL,                     -- bit 7
    uploadState                [8] IMPLICIT Lr_UploadState OPTIONAL,                -- bit 8
    numberOfRelatedObjectsInUse [9] IMPLICIT Unsigned8 OPTIONAL,                   -- bit 9
    listOfRelatedObjects       [10] IMPLICIT SEQUENCE OF Gn_AccessSpecification OPTIONAL, -- bit 10
    contentsSize               [11] IMPLICIT Lr_Size OPTIONAL,                     -- bit 11
    loadImageName              [12] IMPLICIT Gn_Name OPTIONAL,                     -- bit 12
    fixedLengthSegment         [13] IMPLICIT Lr_FixedLengthSegment OPTIONAL,       -- bit 13
    fixedSegmentLength         [14] IMPLICIT Lr_Size OPTIONAL,                     -- bit 14
    intersegmentRequestTimeout [15] IMPLICIT Ar_ServiceTimeOut OPTIONAL,           -- bit 15
    loadRegionSharable         [16] IMPLICIT Boolean OPTIONAL                       -- bit 16
}

```

Mn_NotifierAttrValues

```

Mn_NotifierAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,           -- bit 1
    listOfSupportedAttributes  [2] IMPLICIT Mn_NotifierAttrMap OPTIONAL,             -- bit 2
    listOfOperationalServices  [3] IMPLICIT Mn_NotifierServiceMap OPTIONAL,         -- bit 3
    enabled                    [4] IMPLICIT Gn_Enabled OPTIONAL,                   -- bit 4
    notificationArepClass      [5] IMPLICIT Gn_NumericID OPTIONAL,                 -- bit 5
    notificationArep           [6] IMPLICIT Gn_NumericID OPTIONAL,                 -- bit 6
    notificationType           [7] IMPLICIT Ev_NotificationType OPTIONAL,           -- bit 7
    containedMessageType        [8] IMPLICIT Ev_ContainedMessageType OPTIONAL,       -- bit 8
    containedEventData          [9] IMPLICIT Ev_ContainedEventData OPTIONAL,         -- bit 9
    lastNotificationSequenceNumber [10] IMPLICIT Ev_SequenceNumber OPTIONAL,       -- bit 10
    listOfEvents               [11] IMPLICIT Gn_NumericID OPTIONAL                  -- bit 11
}

```

Mn_VariableAttrValues

```

Mn_VariableAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,           -- bit 1
    listOfSupportedAttributes  [2] IMPLICIT Mn_VariableAttrMap OPTIONAL,            -- bit 2
    listOfOperationalServices  [3] IMPLICIT Mn_VariableServiceMap OPTIONAL,         -- bit 3
    symbolicAddress            [4] IMPLICIT Gn_SymbolicAddress OPTIONAL,           -- bit 4
    dataType                   [5] IMPLICIT Gn_NumericID OPTIONAL,                 -- bit 5
    length                     [6] IMPLICIT Unsigned8 OPTIONAL,                    -- bit 6
    variableLengthConveyance    [7] IMPLICIT Vr_variableLengthConveyance OPTIONAL,  -- bit 7
    numberOfArrayElements      [8] IMPLICIT Unsigned8 OPTIONAL,                    -- bit 8
    accessPrivilege            [9] IMPLICIT Vr_AccessPrivilege OPTIONAL,           -- bit 9
    localAddress               [10] IMPLICIT Gn_NumericAddress OPTIONAL,           -- bit 10
    fullyNestedTypeDescription [11] IMPLICIT Gn_FullyNestedTypeDescription OPTIONAL -- bit 11
}

```

Mn_VariableListAttrValues

```

Mn_VariableListAttrValues ::= SEQUENCE {
    optionalParametersMap      [0] IMPLICIT Gn_OptionalParametersMap8,
    commonAttrValues           [1] IMPLICIT Mn_CommonAttrValues OPTIONAL,           -- bit 1
    listOfSupportedAttributes  [2] IMPLICIT Mn_VariableListAttrMap OPTIONAL,        -- bit 2
    listOfOperationalServices  [3] IMPLICIT Mn_VariableListServiceMap OPTIONAL,     -- bit 3
    numberOfEntries            [4] IMPLICIT Unsigned8 OPTIONAL,                    -- bit 4
    listOfVariables            [5] IMPLICIT SEQUENCE OF Gn_AccessSpecification OPTIONAL, -- bit 5
    deletable                  [6] IMPLICIT Gn_Deletable OPTIONAL,                 -- bit 6
    accessPrivilege            [7] IMPLICIT Vr_AccessPrivilege OPTIONAL             -- bit 7
}

```

4.1.3.15.9.3 Mn_ApoClassAndObjects

```

Mn_ApoClassAndObjects ::= SEQUENCE {
    numericIdentifier          [0] IMPLICIT Gn_NumericID,
    romRamFlag                [1] IMPLICIT Gn_RomRamFlag,
    maxNameLength             [2] IMPLICIT Unsigned8,
    accessProtectionSupported [3] IMPLICIT Gn_AccessProtection,
    versionOfList             [4] IMPLICIT Mn_Version,
    localReference            [5] IMPLICIT Unsigned32,
    optionalParametersMap     [6] IMPLICIT Gn_OptionalParametersMap16 OPTIONAL,
    numericIdentifier          [7] IMPLICIT Gn_NumericID OPTIONAL,      -- bit 1
    numberOfObjects           [8] IMPLICIT Unsigned8 OPTIONAL,          -- bit 2
    localReference            [9] IMPLICIT Unsigned32 OPTIONAL,          -- bit 3
    numericIdentifier         [10] IMPLICIT Gn_NumericID OPTIONAL,       -- bit 4
    numberOfObjects           [11] IMPLICIT Unsigned8 OPTIONAL,          -- bit 5
    localReference            [12] IMPLICIT Unsigned32 OPTIONAL,          -- bit 6
    numericIdentifier         [13] IMPLICIT Gn_NumericID OPTIONAL,       -- bit 7
    numberOfObjects           [14] IMPLICIT Unsigned8 OPTIONAL,          -- bit 8
    localReference            [15] IMPLICIT Unsigned32 OPTIONAL,          -- bit 9
    numericIdentifier         [16] IMPLICIT Gn_NumericID OPTIONAL,       -- bit 10
    numberOfObjects           [17] IMPLICIT Unsigned8 OPTIONAL,          -- bit 11
    localReference            [18] IMPLICIT Unsigned32 OPTIONAL          -- bit 12
}

```

4.1.3.15.9.4 Mn_ArEndpointInfo

```

Mn_ArEndpointInfo ::= SEQUENCE {
    configuredMaxPduSizeSending      [0] IMPLICIT Unsigned8,
    configuredMaxPduSizeReceiving    [1] IMPLICIT Unsigned8,
    configuredMaxOutstandingServicesRequesting [2] IMPLICIT Unsigned8,
    configuredMaxOutstandingServicesResponding [3] IMPLICIT Unsigned8,
    listOfSupportedServices          [4] IMPLICIT Mn_ServiceMap,
    configuredMaxDataStructureNestingLevel [5] IMPLICIT Unsigned8
}

```

4.1.3.15.9.5 Mn_Deletable

```

Mn_Deletable ::= Boolean
-- The value of TRUE means the object is remotely deletable.
-- The value of FALSE means the object is not remotely deletable.

```

4.1.3.15.9.6 Mn_InuseArEndpointInfo

```

Mn_InuseArEndpointInfo ::= SEQUENCE {
    negotiatedMaxPduSizeSending      [0] IMPLICIT Unsigned8,
    negotiatedMaxPduSizeReceiving    [1] IMPLICIT Unsigned8,
    negotiatedMaxOutstandingServicesRequesting [2] IMPLICIT Unsigned8,
    negotiatedMaxOutstandingServicesResponding [3] IMPLICIT Unsigned8,
    outstandingServicesRequestingCounter [4] IMPLICIT Unsigned8,
    outstandingServicesRespondingCounter [5] IMPLICIT Unsigned8,
    negotiatedMaxDataStructureNestingLevel [6] IMPLICIT Unsigned8
}

```

4.1.3.15.9.7 Mn_ManufacturerIdentifier

```

Mn_ManufacturerIdentifier ::= SEQUENCE {
    vendorName      [0] IMPLICIT OctetString,
    modelIdentifier [1] IMPLICIT OctetString,
    vendorRevision  [2] IMPLICIT OctetString,
    serialNumber    [3] IMPLICIT OctetString
}

```

4.1.3.15.9.8 Mn_NetworkAccessState

```

Mn_NetworkAccessState ::= SEQUENCE {
    serviceAccessStatus [0] IMPLICIT Unsigned8 {
        readyForCommunication (1),
        limitedNumberOfServices (2),
        loadingNonInteracting (3),
        loadingInteracting (4)
    },
    operationalStatus [1] IMPLICIT Unsigned8 {
        operational (1),
        partiallyOperational (2),
        notOperational (3),
        needsMaintenance (4)
    }
}

```

4.1.3.15.9.9 Mn_PduSupportedMap

```

Mn_PduSupportedMap ::= BitString {
    getAttributes2PDU          (1),    --All mandatory
    unsolicitedStatusPDU      (2),
    setAttributes2PDU          (3),
    clientPulledDownloadPDU    (4),
    clientPullUploadPDU        (5),
    serverPullDownloadPDU      (6),
    serverPulledUploadPDU      (7),
    createFunctionInvocationPDU (8),
    deleteFunctionInvocationPDU (8),
    startPDU                   (9),
    stopPDU                    (9),
    resumePDU                   (9),
    resetPDU                    (9),
    kill PDU                    (10),
    read2PDU                    (11),
    writePDU                     (12),
    physicalReadPDU             (15),
    physicalWritePDU            (16),
    informationReportPDU        (17),
    createVariableListPDU       (19),
    deleteVariableListPDU      (19),
    eventNotificationPDU        (20),
    confirmedAcknowledgeEventPDU (22),
    enableEventPDU              (23),
    clientPushDownloadPDU       (25),
    getAttributesPDU            (26),    -- Selective
    getAttributesListPDU        (27),
    subscribePDU                (28),
    readListPDU                 (29),
    writeListPDU                 (30),
    informationReportListPDU     (31),
    exchangePDU                 (32),
    unconfirmedAcknowledgeEventsPDU (33),
    getEventSummaryPDU          (34),
    getEventSummaryListPDU      (35),
    queryEventSummaryPDU        (36),
    notificationRecoveryPDU      (37),
    discardPDU                   (38),
    actionInvokePDU              (39),
    actionReturnPDU             (40),
    setAttributes1PDU           (41),
    read1PDU                     (42),
    getAttributes1PDU           (43),
    concludePDU                 (44)
}

```

4.1.3.15.9.10 Mn_SelectedAttributes

```

Mn_SelectedAttributes ::= Unsigned8 {
    allMandatory          (0),    -- All mandatory attributes
    userSpecified         (128),  -- Specified by the AttributeMap
    minimumMandatory      (255)   -- Minimum mandatory attributes
}

```

4.1.3.15.9.11 Mn_ServiceMap

Mn_ActionServiceMap

```

Mn_ActionServiceMap ::= BitString8 {
    actionInvoke          (1),
    actionReturn          (2)
}

```

Mn_ApServiceMap

```

Mn_ApServiceMap ::= BitString8 {
    subscribe              (1),
    identify               (2),
    getStatus              (3),
    statusNotification     (4),
    initiate               (5),
    terminate              (6),
    conclude               (7)
}

```

Mn_EventServiceMap

```

Mn_EventServiceMap ::= BitString8 {
    unconfirmedAcknowledgeEvents (1),
    confirmedAcknowledgeEvents (2),
    enableEvents (3),
    getEventSummary (4),
    getEventSummaryList (5),
    queryEventSummary (6)
}

```

Mn_EventListServiceMap

```

Mn_EventListServiceMap ::= BitString8 {
    getEventSummary (1),
    queryEventSummary (2)
}

```

Mn_FunctionInvocationServiceMap

```

Mn_FunctionInvocationServiceMap ::= BitString8 {
    start (1),
    stop (2),
    resume (3),
    reset (4),
    kill (5)
}

```

Mn_LoadRegionServiceMap

```

Mn_LoadRegionServiceMap ::= BitString8 {
    initiateLoad (1),
    pushSegment (2),
    pullSegment (3),
    terminateLoad (4),
    discard (5)
}

```

Mn_NotifierServiceMap

```

Mn_NotifierServiceMap ::= BitString8 {
    EventNotification (1),
    notificationRecovery (2)
}

```

Mn_VariableListServiceMap

```

Mn_VariableListServiceMap ::= BitString8 {
    read (1),
    write (2),
    readList (3),
    writeList (4),
    exchange (5),
    informationReport (6),
    informationReportList (7)
}

```

Mn_VariableServiceMap

```

Mn_VariableListServiceMap ::= BitString8 {
    read (1),
    write (2),
    readList (3),
    writeList (4),
    exchange (5),
    informationReport (6),
    informationReportList (7)
}

```

4.1.3.15.10 Variable ASE Types

4.1.3.15.10.1 Vr_AccessPrivilege

```
Vr_AccessPrivilege ::= BitString {
    rightToReadPassword          (23),
    rightToWritePassword         (22),
    rightToDeletePassword        (21),
    rightToReadAccessGroup       (19),
    rightToWriteAccessGroup       (18),
    rightToDeleteAccessGroup      (17),
    rightToReadAllPartners       (31),
    rightToWriteAllPartners       (30),
    rightToDeleteAllPartners      (29),
    password_Bit1                (7),
    password_Bit2                (6),
    password_Bit3                (5),
    password_Bit4                (4),
    password_Bit5                (3),
    password_Bit6                (2),
    password_Bit7                (1),
    password_Bit8                (0),
    access_Groups-1              (15),
    access_Groups-2              (14),
    access_Groups-3              (13),
    access_Groups-4              (12),
    access_Groups-5              (11),
    access_Groups-6              (10),
    access_Groups-7              (9),
    access_Groups-8              (8)
}
```

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

4.1.3.15.10.2 Vr_VariableLengthConveyance

Vr_VariableLengthConveyance ::= Boolean

- True means only the octets that are in use in a string are conveyed.
- False means all the octets in a string are conveyed.

4.1.3.15.10.3 Vr_WriteResult

```
Vr_WriteResult ::= CHOICE {
    success          [0] IMPLICIT NULL,
    service          [5] IMPLICIT Er_Service,
    access           [6] IMPLICIT Er_Access,
    others           [8] IMPLICIT Er_Other
}
```

4.1.4 Data Types

4.1.4.1 Notation for the Boolean Type

Boolean ::= BOOLEAN

- TRUE if the value is non-zero.
- FALSE if the value is zero.

4.1.4.2 Notation for the Integer Type

Integer ::= INTEGER -- any integer

Integer8 ::= INTEGER (-128..+127) -- range $-2^7 \leq i \leq 2^7-1$

Integer16 ::= INTEGER (-32768..+32767) -- range $-2^{15} \leq i \leq 2^{15}-1$

Integer32 ::= INTEGER -- range $-2^{31} \leq i \leq 2^{31}-1$

4.1.4.3 Notation for the Unsigned Type

Unsigned ::= INTEGER -- any non-negative integer

Unsigned8 ::= INTEGER (0..255) -- range $0 \leq i \leq 2^8-1$

Unsigned16 ::= INTEGER (0..65535) -- range $0 \leq i \leq 2^{16}-1$

Unsigned32 ::= INTEGER -- range $0 \leq i \leq 2^{32}-1$

4.1.4.4 Notation for the Floating Point Type

Floating32 ::= BIT STRING SIZE (4) -- IEEE-754 Single precision

Floating64 ::= BIT STRING SIZE (8) -- IEEE-754 Double precision

4.1.4.5 Notation for the BitString Type

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

4.1.4.6 Notation for the OctetString Type

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

4.1.4.7 Notation for VisibleString Type

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

4.1.4.8 Notation for the UNICODEString Type

UNICODEString ::= UNICODEString -- 16-bit character code set defined in ISO 10646.

4.1.4.9 Notation for the FieldbusTime Type

FieldbusTime -- Format and granularity are defined in IEC 61158-3 and IEC 61158-4.

4.1.4.10 Notation for the Universal Time Type

UniversalTime ::= VisibleString SIZE (12) -- YYYYMMDDhhmmss

4.1.4.11 Notation for Binary Time Type

BinaryTime0 ::= BIT STRING SIZE (16) -- 10 µs resolution
 BinaryTime1 ::= BIT STRING SIZE (16) -- .1 ms resolution
 BinaryTime2 ::= BIT STRING SIZE (16) -- 1 ms resolution
 BinaryTime3 ::= BIT STRING SIZE (16) -- 10 ms resolution
 BinaryTime4 ::= BIT STRING SIZE (32) -- 10 µs resolution
 BinaryTime5 ::= BIT STRING SIZE (32) -- 1 ms resolution
 BinaryTime6 ::= BIT STRING SIZE (32) -- 1 ms resolution
 BinaryTime7 ::= BIT STRING SIZE (32) -- 10 ms resolution
 BinaryTime8 ::= BIT STRING SIZE (48) -- 10 µs resolution
 BinaryTime9 ::= BIT STRING SIZE (48) -- .1 ms resolution

4.1.4.12 Notation for BCD Type

BCD ::= Unsigned8 (0..9) -- Lower four bits are used to express one BCD value.

4.1.4.13 Notation for Compact Boolean Array Type

CompactBooleanArray ::= BitString -- Each zero bit representing Boolean value FALSE.
 -- Each one bit representing Boolean value TRUE.
 -- Unused bits, if any, shall be placed in bits 6-0 of the last octet.

4.1.4.14 Notation for Compact BCD Array Type

CompactBCDArray ::= OctetString -- One BCD value is represented by four bits, an unused
 -- nibble, if any, shall be placed in bits 3-0 of the last octet,
 -- and shall be set to 1111.

4.1.4.15 Notation for BinaryDate Type

BinaryDate ::= OctetString7

4.1.4.16 Notation for TimeOfDay Type

TimeOfDay ::= OctetString6

4.1.4.17 Notation for TimeDifference Type

TimeDifference ::= OctetString6

4.1.4.18 Notation for TimeValue Type

TimeValue ::= OctetString8

4.1.5 Reason Codes

4.1.5.1 Introduction

This section specifies the values of the Reason parameter of the Data Link Layer DL-Disconnect service that are supplied by the FAL.

Table 11 – Reason Codes

FAL Reason parameter	Value
Invalid FAL-PDU	31 (Hex)
Remote Address Mismatch	32 (Hex)
Multiple Initiators	33 (Hex)
Invalid DL-Event	34 (Hex)
AREP Busy	35 (Hex)
AREP Not Found	36 (Hex)
Invalid AREP Type	37 (Hex)
DLCEP Not Found	38 (Hex)
Invalid DI PDU	39 (Hex)
Not allowed AREP primitive in connection establishment	3A (Hex)
Number of parallel services exceeded	3B (Hex)
Not allowed service as server	3C (Hex)
Not allowed service as client	3D (Hex)
Context Check Negative	3E (Hex)
Invalid FAL-PDU	3F (Hex)
Invalid Event for Role	40 (Hex)
UCA_AckPDU received and UCC=0	41 (Hex)
RCTimer expired	42 (Hex)
RSTimer expired	43 (Hex)
MCTimer expired	44 (Hex)
MSTimer expired	45 (Hex)
Execution error in cyclic data transfer	46 (Hex)

4.2 Transfer Syntaxes

4.2.1 Transfer Syntax 1

4.2.1.1 FAL Header

4.2.1.1.1 Introduction

All the FAL PDUs shall have the common PDU-header called FalArHeader. The FalArHeader identifies abstract syntax, transfer syntax, and each of the PDUs. This section defines how this header shall be used.

NOTE The structure of the FalArHeader is defined in the main text of this standard.

FAL Protocol Specifier (Bit 8) is always zero (0) for the protocol defined by this standard.

Table 12 – FAL Header

Bit position of the FalArHeader			Abstract Syntax	Encoding Rule	PDU Type	Revision
8	7 6 5 4 3	2 1				
0	0 0 0 0 0	0 0	FAL AS2	TER	Establish-RequestPDU	Revision1
0	0 0 0 0 0	0 1	FAL AS2	TER	Establish-ResponsePDU	Revision1
0	0 0 0 0 0	1 0	FAL AS2	TER	Establish-ErrorPDU	Revision1
0	0 0 0 0 0	1 1	FAL AS2	TER	AbortPDU	Revision1
0	0 0 0 0 1	0 0	FAL AS2	TER	ConfirmedSend-RequestPDU	Revision1
0	0 0 0 0 1	0 1	FAL AS2	TER	ConfirmedSend-ResponsePDU	Revision1
0	0 0 0 0 1	1 0	FAL AS2	TER	UnconfirmedSendPDU	Revision1
0	1 0 0 0 0	0 0	FAL AS1	CER	Establish-CommandPDU	Revision1
0	1 0 0 0 0	0 1	FAL AS1	CER	Establish-AffirmativePDU	Revision1
0	1 0 0 0 0	1 0	FAL AS1	CER	Establish-NegativePDU	Revision1
0	1 0 0 0 0	1 1	FAL AS1	CER	AbortPDU	Revision1
0	1 0 0 0 1	0 0	FAL AS1	CER	ConfirmedSend-CommandPDU	Revision1
0	1 0 0 0 1	0 1	FAL AS1	CER	ConfirmedSend-AffirmativePDU	Revision1
0	1 0 0 0 1	1 0	FAL AS1	CER	ConfirmedSend-NegativePDU	Revision1
0	1 0 0 0 1	1 1	FAL AS1	CER	UnconfirmedSend-CommandPDU	Revision1
0	1 0 0 1 0	0 0	FAL AS1	CER	UnconfirmedAcknowledgedSend-CommandPDU	Revision1
0	1 0 0 1 0	0 1	FAL AS1	CER	UnconfirmedAcknowledgedSend-AffirmativePDU	Revision1
0	1 0 0 1 0	1 0	FAL AS1	CER	IdleSend-CommandPDU	Revision1
0	1 0 0 1 0	1 1	FAL AS1	CER	AR-XON-OFF-CommandPDU	Revision1
0	0 1 0 S 0	0 0	FAL AS1	MER	Establish-CommandPDU	Revision1
0	0 1 0 S 0	0 1	FAL AS1	MER	Establish-AffirmativePDU	Revision1
0	0 1 0 S 0	1 0	FAL AS1	MER	Establish-NegativePDU	Revision1
0	0 1 0 S 0	1 1	FAL AS1	MER	AbortPDU	Revision1
0	0 1 0 S 1	0 0	FAL AS1	MER	ConfirmedSend-CommandPDU	Revision1
0	0 1 0 S 1	0 1	FAL AS1	MER	ConfirmedSend-AffirmativePDU	Revision1
0	0 1 0 S 1	1 0	FAL AS1	MER	ConfirmedSend-NegativePDU	Revision1
0	0 1 0 S 1	1 1	FAL AS1	MER	UnconfirmedSend-CommandPDU	Revision1
0	1 1 0 S 0	0 0	FAL AS1	BER	BufferedDataTransferPDU	Revision1
0	0 0 1 0 0	0 0	FAL AS2	TER	UnconfirmedAcknowledgedSend-CommandPDU	Revision1
0	0 0 1 0 0	0 1	FAL AS2	TER	UnconfirmedAcknowledgedSend-AffirmativePDU	Revision1
0	0 0 1 0 0	1 0	FAL AS2	TER	IdleSend-CommandPDU	Revision1
0	0 0 1 0 0	1 1	FAL AS2	TER	AR-XON-OFF-CommandPDU	Revision1
0	0 0 1 0 1	0 0	FAL AS2	TER	Establish-Request2PDU	Revision1
1	X X X X X	X X			Reserved	

-- All other code points are reserved for additional protocols and future revisions.

-- S-bit (bit4) is used for segmentation purposes only. It indicates to the receiver of an FAL PDU, if this PDU is the last segment (S = 0) or not (S = 1).

4.2.1.2 Traditional Encoding Rule (TER)

4.2.1.2.1 Traditional Encoding Rule (TER)

4.2.1.2.1.1 Introduction

The Traditional Encoding Rule (TER) is a preferable encoding rule that is compatible with existing standards.

4.2.1.2.1.2 TER Descriptions

Overview of Encoding

The FAL-PDUs encoded with the TER shall have a uniform format. The FAL-PDUs shall consist of two major parts, the “APDU Header” part and the “APDU Body” part as shown in the figure below:

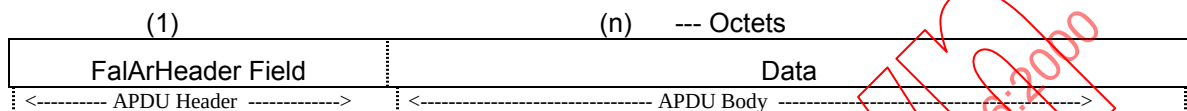


Figure 6 – APDU Overview

APDU Header Encoding

The APDU Header part is always present in all APDUs which conform to this standard. It consists of one field: the FalArHeader Field. Refer to 4.2.1.1 for the encoding rule of the FalArHeader field.

APDU Body Encoding

The FAL-PDUs are encoded with the TER in the following manner.

```
FAL-PDU ::= {Components}
Components ::= UserData ||
               IdentificationInformation ContentsOctets
IdentificationInformation ::= P/C Flag Tag Length
ContentsOctets ::= OCTETSTRING
```

Structure of the Identification Information

The Identification Information consists of the P/C flag, the Tag field, and the Length field as shown in the following figure:

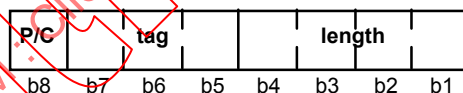


Figure 7 – Identification Information (format 1)

The P/C flag indicates either the ContentsOctets is a simple component (primitive types, such as Integer8), or a structured component (constructed, such as SEQUENCE, SEQUENCE OF types).

P/C Flag =0 means the ContentsOctets is a simple component.
P/C Flag =1 means the ContentsOctets is a structured component.

The Tag field identifies the semantics of the ContentsOctets.

The Length field indicates the length of the ContentsOctets in octets if it is a primitive type and the number of contained components if it is a structured type.

The three components of the Identification Information are allotted in one octet as follows:

P/C flag bit8
Tag field bit7 - bit5, where bit7 is the msb and bit5 is the lsb.
Length field bit3 - bit1, where bit4 is the msb and bit1 is the lsb.

If the number of the Tag is greater than six, or the number of the Length is greater than 14, the extension octet for each of the fields shall be used.

If the extension octet for the Tag is used, the Tag field in the Identification Information shall be set to seven. The Tag Extension octet that immediately follows the Identification Information shall contain the number of the Tag, with bit8 of the extension octet as the msb and bit1 the lsb as shown below:



Figure 8 – Identification Information (format 2)

If the extension octet for the Length is used, the Length field in the Identification Information shall be set to 15. The Length Extension octet that immediately follows the Identification Information shall contain the number of the Length, with bit8 of the extension octet as the msb and bit1 the lsb as shown below:



Figure 9 – Identification Information (format 3)

If both the Tag and the Length fields are extended, the Tag Extension octet follows the Identification Information and is followed by the Length Extension octet as shown below:

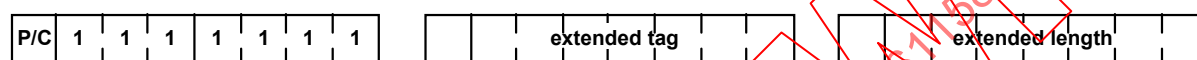


Figure 10 – Identification Information (format 4)

If the abstract syntax does not contain a tag for a data item, then no Identification Information is encoded. The semantics and length of this data item are implicitly known by the abstract syntax.

Encoding of Simple Variables

Encoding of a Boolean Value

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

Encoding of an Integer Value

- 1) The encoding of a fixed-length Integer value of Integer8, Integer16, and Integer32 types shall be primitive, and the ContentsOctets shall consist of exactly one, two, or four octets, respectively.
- 2) The ContentsOctets shall be a two's complement binary number equal to the integer value, and consist of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 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 1 of the last octet as bit zero and ending the numbering with bit 8 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 8 of the first octet, and then reducing this value by the numerical value assigned to bit 8 of the first octet if that bit is set to one.

Encoding of an Unsigned Value

- 1) The encoding of a fixed-length Unsigned value of Unsigned8, Unsigned16, and Unsigned32 types shall be primitive, and the ContentsOctets shall consist of exactly one, two, or four octets, respectively.
- 2) The ContentsOctets shall be a binary number equal to the Unsigned value, and consist of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 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 1 of the last octet as bit zero and ending the numbering with bit 8 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.

Encoding of a Floating-Point Value

- 1) The encoding of a Floating32 or Floating64 value shall be primitive, and the ContentsOctets shall consist of exactly four or eight octets, respectively.
- 2) The ContentsOctets shall contain floating-point values defined in conformance with ANSI/IEEE Standard 754. The sign is encoded in bit 8 of the first octet. It is followed by the exponent starting from bit 7 of the first octet, and then the mantissa starting from bit 7 of the second octet for Floating32 and from bit 4 of the second octet for Floating64.

Encoding of a Visible String Value

- 1) The encoding of a variable length VisibleString value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of octets in the VisibleString value.
 - 2.1) If the VisibleString is of zero length, there shall be no ContentsOctets, and the Length shall be zero.
- 3) If the VisibleString is not of zero length, the ContentsOctets shall be equal in value to the octets in the data value, in the order they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the ContentsOctets.

Encoding of an Octet String Value

- 1) The encoding of a variable length OctetString value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of octets in the OctetString value.
 - 2.1) If the OctetString is of zero length, there shall be no ContentsOctets, and the Length shall be zero.
- 3) If the OctetString is not of zero length, the ContentsOctets shall be equal in value to the octets in the data value, in the order they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the ContentsOctets.

Encoding of a BinaryDate Value

- 1) The encoding of a BinaryDate value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of octets in the BinaryDate value.
- 3) The ContentsOctets shall be equal in value to the octets in the data value, as shown in the figure below:

		msb				lsb				
Bit-No		7	6	5	4	3	2	1	0	
Octet										
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		RS V	RS V	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	0...59 min
4		SU	RS V	RS V	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	0...23 hours
5		day of week 2 ² 2 ¹ 2 ⁰			day of month 2 ⁴ 2 ³ 2 ² 2 ¹ 2 ⁰				1...7 day of week 1...31 day of month	
6		RS V	RS V	2 ⁵	2 ⁴	2 ³	2 ²	2 ¹	2 ⁰	1...12 months
7		RS V	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

Figure 11 - Coding of the Data Type BinaryDate

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).

Encoding of a Time Of Day Value

- 1) The encoding of a Time Of Day value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of octets in the Time Of Day value.
- 3) The ContentsOctets shall be equal in value to the octets in the data value, as shown in the figure below:

bits	8	7	6	5	4	3	2	1	
octets									
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	number of milliseconds 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	number of days since 01.01.84
5	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	
6	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
MSB									

Figure 12 – Encoding of Time Of Day Value**Encoding of a Time Difference Value**

- 1) The encoding of a Time Difference value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of octets in the Time Difference value.
- 3) The ContentsOctets shall be equal in value to the octets in the data value, as shown in the figure below:

bits	8	7	6	5	4	3	2	1	
octets									
1	0	0	0	0	2^{27}	2^{26}	2^{25}	2^{24}	number of milliseconds
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
6	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
MSB									

Figure 13 – Encoding of Time Difference Value

Encoding of a Bit String Value

- 1) The encoding of a variable length BitString value shall be primitive.
- 2) The Length field shall indicate the number of octets in the BitString encoding, which is the number of octets used to encode a BitString.
 - 2.1) If the BitString is empty (no BitString value at all), there shall be no ContentsOctets, and the Length field shall be zero.
- 3) If the BitString is not empty, the ContentsOctets shall encode the BitString value as follows:
 - 3.1) The value of the BitString, commencing with the first bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet up to and including the last octet of the ContentsOctets.

Encoding of a Time Value

- 1) The encoding of a Time value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of octets in the Time value.
- 3) The ContentsOctets shall be equal in value to the octets in the data value, as shown in the figure below:

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

Figure 14 – Encoding of TimeValue

Encoding of a Null Value

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

Encoding of an ANY Value

The Data Type ANY contains one or more data elements of the Data Types which are chained together without any gap. The composition is known implicitly.

Encoding of Structured Types

When a structured type is encoded, the P/C flag of the Identification Information shall be set to one. The Length field, whether or not it is extended, shall contain the number of components of the structured type being encoded.

Encoding of a SEQUENCE Value

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

A SEQUENCE type value may contain a simple variable or a further structured variable as its components. If a SEQUENCE type contains another structured type value, it shall be counted as a single component even if it contains several components.

Encoding of a SEQUENCE OF Value

The SEQUENCE OF type represents a succession of components. It is comparable to an array.

A SEQUENCE OF type value may contain one or more simple or constructed variables. If a SEQUENCE OF type contains another structured type value, it shall be counted as a single component even if it contains several components.

The encoding is as for the structure SEQUENCE. For the statement of the number of components the number of repetitions shall be taken into account.

Encoding of a CHOICE Value

A CHOICE type represents a selection from a set of predefined values. The components of a CHOICE construct shall have different tags to allow proper identification. Instead of the CHOICE construct, the actually selected component is encoded. Only one component shall be encoded for a CHOICE.

Object Definition Parameter

The mapping of the individual object definitions onto the parameters "List Of Objects and Attributes" (data type Gn_ObjectDefinition) of the GetAttributes2 and SetAttributes2 PDUs is shown in this clause.

NOTE The semantics of the Object Definition Parameter of type Gn_ObjectDefinition is application specific and the FAL does not encode or decode it. However, since prior standards defined standard semantics of this parameter, and it is used with the TER, its definition is placed here. Readers are advised that this clause is not for the FAL, but for the FAL user.

The object class is the identifier of the object and indicates the class to which this object belongs. The other object attributes are object specific and shall be coded as a string of octets. In addition to the object attributes, the object class shall be transmitted in the GetAttributes and SetAttributes services, except List Header, whose numeric ID is zero.

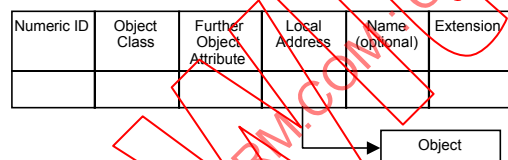


Figure 15 – Structure of an Object Definition

4.2.1.2.2 Object Definition

```
Object-Definition ::= ListHeader
                    || DataTypeList
                    || StaticList
                    || VariableListDefinition
                    || FunctionInvocationDefinition
```


4.2.1.2.2.1 ListHeader

```
ListHeader ::= ANY {
    Unsigned16,
    Boolean,
    Unsigned8,
    Boolean,
    Integer16,
    Unsigned32,
    Unsigned16,
    Unsigned32,
    Unsigned16,
    Unsigned16,
    Unsigned32,
    Unsigned16,
    Unsigned16,
    Unsigned16,
    Unsigned32,
    Unsigned16,
    Unsigned16,
    Unsigned32
}
-- numericId
-- romRamFlag
-- maxNameLength
-- accessProtectionSupported
-- versionOfObjectDefinition
-- localReferenceOfListHeader
-- numberOfEntriesInDataTypeList
-- localReferenceOfDataTypeList
-- firstNumericIdOfStaticList
-- numberOfEntriesInStaticList
-- localReferenceOfStaticList
-- firstNumericIdOfVariableListDefinition
-- numberOfEntriesInVariableListDefinition
-- localReferenceOfVariableListDefinition
-- firstNumericIdOfFunctionInvocationDefinition
-- numberOfEntriesInFunctionInvocationDefinition
-- localReferenceOfFunctionInvocationDefinition
```

4.2.1.2.2.2 DataTypeList

```
DataTypeList ::= DataTypeDefinition
|| StructuredDataTypeDefinition
DataTypeDefinition ::= ANY {
    Unsigned16,
    Integer8,
    Unsigned8,
    VisibleString
}
-- numericId
-- objectClass
-- dataTypeNameLength
-- dataTypeName
StructuredDataTypeDefinition ::= ANY {
    Unsigned16,
    Integer8,
    Unsigned8,
    recordList SEQUENCE OF {
        Unsigned16,
        Unsigned8,
    }
}
-- numericId
-- objectClass
-- numberOfElements
-- numericIdOfDataTypeDefinition
-- dataLength
```

4.2.1.2.2.3 StaticList

```
StaticList ::= VariableDefinition
|| ArrayDefinition
|| StructureDefinition
|| LoadRegionDefinition
|| EventDefinition
VariableDefinition ::= ANY {
    Unsigned16,
    Integer8,
    Unsigned16,
    Unsigned8,
    Vr_AccessPrivilege,
    Unsigned32,
    VisibleString,
    Unsigned8,
    OctetString
}
-- numericId
-- objectClass
-- numericIdOfDataTypeDefinition
-- dataLength
-- accessPrivilege
-- localReferenceOfVariable
-- variableName
-- Length of the extension field. If the extension field is not present, the value of
  this field shall be zero.
-- extension
ArrayDefinition ::= ANY {
    Unsigned16,
    Integer8,
    Unsigned16,
    Unsigned8,
    Unsigned8,
    Vr_AccessPrivilege,
    Unsigned32,
    VisibleString,
    Unsigned8,
    OctetString
}
-- numericId
-- objectClass
-- numericIdOfDataTypeDefinition
-- dataLength
-- numberOfElements
-- accessPrivilege
-- localReferenceOfArray
-- arrayName
-- Length of the extension field. If the extension field is not present, the value of
  this field shall be zero.
-- extension
```

```

StructureDefinition ::= ANY {
    Unsigned16, -- numericId
    Integer8, -- objectClass
    Unsigned16, -- numericIdOfDataTypeDefinition
    Vr_AccessPrivilege, -- accessPrivilege
    VisibleString, -- structureName
    Unsigned8, -- Length of the extension field. If the extension field is not present, the value of
    -- this field shall be zero.
    OctetString, -- extension
    SEQUENCE OF Unsigned32 -- localReferenceOfElement
}

```

```

LoadRegionDefinition ::= ANY {
    Unsigned16, -- numericId
    Integer8, -- objectClass
    Unsigned16, -- loadRegionSize
    Vr_AccessPrivilege, -- accessPrivilege
    Unsigned32, -- localReferenceOfLoadRegion
    Unsigned8, -- loadRegionState
    Unsigned8, -- uploadState
    Integer16, -- numberOfRelatedObjectsInUse
    VisibleString, -- loadRegionName
    Unsigned8, -- Length of the extension field. If the extension field is not present, the value of
    -- this field shall be zero.
    OctetString -- extension
}

```

```

EventDefinition ::= ANY {
    Unsigned16, -- numericId
    Integer8, -- objectClass
    Unsigned16, -- numericIdOfEventData
    Unsigned8, -- eventDataLength
    Vr_AccessPrivilege, -- accessPrivilege
    Boolean, -- enabled
    VisibleString, -- eventName
    Unsigned8, -- Length of the extension field. If the extension field is not present, the value of
    -- this field shall be zero.
    OctetString -- extension
}

```

4.2.1.2.2.4 VariableListDefinition

```

VariableListDefinition ::= ANY {
    Unsigned16, -- numericId
    Integer8, -- objectClass
    Unsigned16, -- numericIdOfElements
    Vr_AccessPrivilege, -- accessPrivilege
    Boolean, -- deletable
    SEQUENCE OF Unsigned32,
    VisibleString, -- variableListName
    Unsigned8, -- Length of the extension field. If the extension field is not present, the value of
    -- this field shall be zero.
    OctetString -- extension
}

```

4.2.1.2.2.5 FunctionInvocationDefinition

```

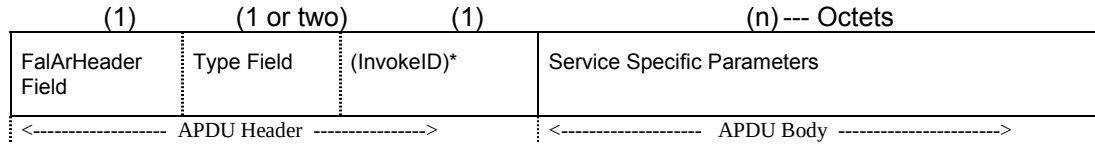
FunctionInvocationDefinition ::= ANY {
    Unsigned16, -- numericId
    Integer8, -- objectClass
    Unsigned8, -- numberOfRelatedObjects
    Fi_AccessPrivilege, -- accessPrivilege
    Boolean, -- deletable
    Boolean, -- reusable
    Unsigned8, -- functionInvocationState
    SEQUENCE OF Unsigned16, -- NumericIdOfLoadRegion
    VisibleString, -- functionInvocationName
    Unsigned8, -- Length of the extension field. If the extension field is not present, the value of
    -- this field shall be zero.
    OctetString -- extension
}

```

4.2.1.3 Compact Encoding Rule (CER)

4.2.1.3.1 Introduction

The APDUs that conform to this standard shall be encoded in a uniform format as shown in the figure below. The APDUs consist of two major parts: the "APDU Header" part and the "APDU Body" part.



NOTE The presence of the InvokeID Field depends on the APDU type.

Figure 16 – APDU Overview

In order to realize an efficient APDU while maintaining flexible encoding, different encoding rules are used for the APDU Header part and the APDU Body part.

NOTE The IEC Data Link layer service provides a DLSDU parameter which implies the length of the APDU. Hence, the APDU length information is not included in the APDU.

4.2.1.3.2 APDU Header encoding

The APDU Header part is always present in all APDUs which conform to this standard. It consists of three fields: the AR Control Field, the Type Field, and the optional InvokeID Field. They are shown in Figure 16.

4.2.1.3.2.1 APDU Header encoding

Refer to 4.2.1.1 for the encoding rule of the FalArHeader field.

4.2.1.3.2.2 Encoding of Type Field

- 1) Both the APDU type and the service type are encoded in the Type Field that is always the second of the APDUs.
- 2) All bits of the Type Field are used to encode the service type.
 - 2.1) The service types are encoded in bits 8 to 1 of the Type Field, with bit 8 the most significant bit and bit 1 the least significant bit. The range of the service type shall be between 0 (zero) and 254, inclusive.
 - 2.2) The value of 255 is reserved for future extensions to this standard.
 - 2.3) The service types are specified in the abstract syntax as a positive integer value. This positive integer value shall be encoded in the bits described in 2.1), where the most significant bit of a positive integer value shall be aligned to bit 8 and its least significant bit to bit 1.
- 3) Figure 17 illustrates the encoding of the Type Field.

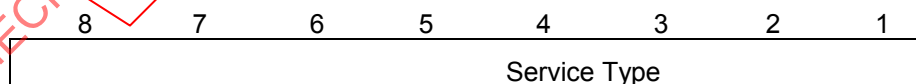


Figure 17 – Type Field

4.2.1.3.2.3 Encoding of InvokeID Field

- 1) The InvokeID Field shall be present if it is indicated in the abstract syntax. Otherwise, this field shall not be present. If present, the Invoke ID parameter supplied by a service primitive shall be placed in this field.

4.2.1.3.2.4 APDU Body Encoding

The FAL encoding rules are based on terms and conventions defined in ISO/IEC 8825. The encoding consists of three components in the following order:

- An Identifier Octet
- Length Octet(s)

- The Contents Octets

The Identifier Octet or Length Octet(s), or both, may be removed from the encoding, depending data type of the field of the APDUs being encoded. The rules for removal are defined later.

Identifier Octet

- 1) The Identifier Octet shall encode the tag defined in the FAL Abstract Syntax and shall consist of one octet.
 - 1.1) The Identifier Octet comprises a class and a number. The class encodes the class of the FAL tag, whereas a number encodes a number associated to each tag.
- 2) Two classes are defined for the Identifier Octet: The Context-Specific class and the FAL-Specific class.
 - 2.1) The scope of a Context-Specific class is limited to the construction in which it is used. The same tag may be used to represent different types in different productions.
 - 2.2) The scope of an FAL-Specific identifier is global to the module definitions of the Abstract Syntax definition. The same tag shall designate the same type within the same module.
 - 2.3) Bit 8 of the Identifier Octets shall indicate its class. If it is set to zero (0), the class shall be Context-Specific. If it is set to one (1), the class shall be FAL-Specific.
- 3) The bits 7 to 1 of the Identifier Octet shall represent the value of the tag, with bit 7 being the most significant bit and bit 1 the least significant bit. This gives tags with the possible values of 0 (zero) through 126 inclusive, for each of the classes.
- 4) The value of 127 is reserved for future extensions this standard.
- 5) The convention for expressing an FAL-specific identifier with a tagged value of two shall be FAL-Specific 2.
- 6) When necessary, the abstract syntax shall explicitly specify which class shall be used. The default class shall be "Context Specific."
- 7) Figure 18 and Figure 19 depict encoding examples of Identifier Octets.

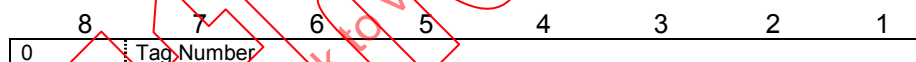


Figure 18 – Identifier Octet (Context-Specific)

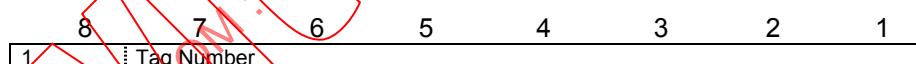


Figure 19 – Identifier Octet (FAL-Specific)

Length Octet(s)

- 1) The Length Octet(s) shall consist of one octet or three octets.
 - 1.1) If the value of the first Length Octet is other than 255, there shall be no subsequent Length Octets and the first octet shall contain the value for the Length Octet defined later.
 - 1.2) If the value of the first Length Octet is 255, there shall be two subsequent Length Octets that shall contain the value for the Length Octets defined later. In this case, the length information of the Contents Octets shall be represented by the last two octets of the Length Octets, where the most significant bit of the second of three Length Octet(s) shall be the most significant bit of the length value and the least significant bit of the third of the three Length Octet(s) shall be the least significant bit of the length value.
- 2) It shall be the sender's option to use either of the Length Octet(s) formats. The three octets format may be used to convey a length value of one, for instance.
- 3) The meaning of the Length Octet(s) depends on the type of the value being encoded. If the encoding of the Contents Octets is primitive, the Length Octet(s) shall contain the number of octets in the Contents Octets. If the encoding of the Contents Octets is constructed, the Length Octet(s) shall contain the number of the first level components of the Contents Octets.

- 4) Figure 20 and Figure 21 depict encoding examples of the Length Octet(s):

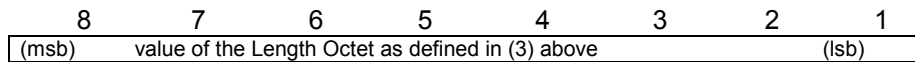


Figure 20 – Length Octet (one-octet format)

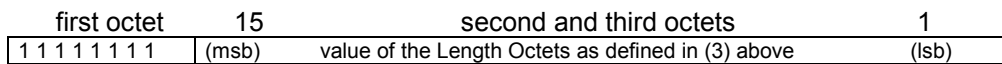


Figure 21 – Length Octet (three-octet format)

Contents Octets

- 1) The Contents Octets shall encode the data value according to the encoding rule defined for its type.
- 2) The Contents Octets shall have either of the following two forms: a primitive encoding or a constructed encoding.
 - 2.1) If the Contents Octets contain a primitive encoding, they represent an encoding of one value.
 - 2.2) If the Contents Octets contain a constructed encoding, they represent an enumerated encoding of more than one value.

4.2.1.3.3 Data Type Encoding Rules (Base Encoding)

The base encoding rules of the following primitive data types defined in IEC 61158-5 are specified in this subclause:

- Boolean
- NULL
- Integer, Integer8, Integer16, Integer32
- Unsigned, Unsigned8, Unsigned16, Unsigned32
- Floating32, Floating64
- BitString, BitString8, BitString16, BitString32
- OctetString, OctetString2, OctetString4, OctetString8, OctetString16
- VisibleString, VisibleString2, VisibleString4, VisibleString8, VisibleString16
- ISO10646String
- UniversalTime
- BinaryTime0, BinaryTime1, BinaryTime2, BinaryTime3, BinaryTime4
- BinaryTime5, BinaryTime6, BinaryTime7, BinaryTime8, BinaryTime9
- CompactBCDArray
- BCD
- FieldbusTime
- CompactBooleanArray
- ANY

The base encoding rules of the following constructed data types are also provided:

- Array
- Structure

4.2.1.3.3.1 Encoding of a NULL Value

- 1) The encoding of a NULL value shall be primitive.

- 2) The Identifier Octet shall be present, and the class number of the NULL value shall be FAL-Specific 2.
- 3) The Length Octet(s) shall be omitted.
- 4) The Contents Octet shall be omitted.
- 5) The NULL type shall not be used to represent that there is no data value present. It may be used in the following two cases:
 - a) An APDU which has no associated parameters:
 Example: NoEffectReqPDU ::= NULL
 - b) In the CHOICE type

4.2.1.3.3.2 Encoding of a Boolean Value

- 1) The encoding of a Boolean value shall be primitive.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) If a Boolean parameter is mandatory in an abstract syntax, then a fixed-length BitString value shall be placed where the Boolean parameter is first present in the abstract syntax and the value of the Boolean parameter and all the subsequent mandatory Boolean parameters shall be encoded in the BitString value in the order they appear. The first Boolean value shall be placed in the most significant bit of the BitString value and the following Boolean values shall be placed in the descending bits, consequently.
- 4) If a Boolean parameter is optional in an abstract syntax, then its value shall be encoded in the next lower order bit of the OptionalParametersMap parameter whose higher order bit is used to indicate the presence of this parameter.
- 5) If the Boolean value is FALSE, the corresponding bit shall be 0 (zero). If the Boolean value is TRUE, the bit shall be 1 (one).

4.2.1.3.3.3 Encoding of a Variable-Length Integer Value

- 1) The encoding of a variable-length Integer value shall be primitive.
- 2) The Identifier Octet shall not be present.
- 3) The Length Octet(s) shall indicate the number of octets in the Contents Octets that are actually encoded.
- 4) The Contents Octets shall be a two's complement binary number equal to the integer value, and consist of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn up to and including the last octet of the Contents Octets.

NOTE The value of a two's complement binary number is derived by numbering the bits in the Contents Octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 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 8 of the first octet, and then reducing this value by the numerical value assigned to bit 8 of the first octet if that bit is set to one.

- 4.1) If the Contents Octets of a variable length Integer value encoding consist of more than one octet, and if the bits of the first octet and bit 8 of the second octet are all ones or zeros, it shall be the sender's option to remove the first octet or not. This process may be repeated to eliminate all such octets. The receivers shall be able to decode the value whether or not such an octet(s) is removed.

4.2.1.3.3.4 Encoding of a Fixed-Length Integer Value

- 1) The encoding of a fixed-length Integer value of Integer8, Integer16, and Integer32 types shall be primitive, and the Contents Octets shall consist of exactly one, two, or four octets, respectively.
- 2) The Identifier Octet and Length Octet(s) shall not be present.

- 3) The Contents Octets shall be a two's complement binary number equal to the integer value, and consist of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn up to and including the last octet of the Contents Octets.

NOTE The value of a two's complement binary number is derived by numbering the bits in the Contents Octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 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 8 of the first octet, and then reducing this value by the numerical value assigned to bit 8 of the first octet if that bit is set to one.

4.2.1.3.3.5 Encoding of a Variable-Length Unsigned Value

- 1) The encoding of a variable-length Unsigned value shall be primitive.
- 2) The Identifier Octet shall not be present.
- 3) The Length Octet(s) shall indicate the number of octets in the Contents Octets that are actually encoded.
- 4) The Contents Octets shall be a binary number equal to the Unsigned value, and consisting of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn up to and including the last octet of the Contents Octets.

NOTE The value of a binary number is derived by numbering the bits in the Contents Octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 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.

- 4.1) If the Contents Octets of an Unsigned value encoding consist of more than one octet, and if the bits of the first octet are all zeros, it shall be the sender's option to remove the first octet or not. This process may be repeated to eliminate such octets. The receivers shall be able to decode the value whether or not such an octet(s) is removed.

4.2.1.3.3.6 Encoding of a Fixed-Length Unsigned Value

- 1) The encoding of a fixed-length Unsigned value of Unsigned8, Unsigned16, and Unsigned32 types shall be primitive, and the Contents Octets shall consist of exactly one, two, or four octets, respectively.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) The Contents Octets shall be a binary number equal to the Unsigned value, and consist of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn up to and including the last octet of the Contents Octets.

NOTE The value of a binary number is derived by numbering the bits in the Contents Octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 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.

4.2.1.3.3.7 Encoding of a Floating Point Value

- 1) The encoding of a Floating32 or Floating64 value shall be primitive, and the Contents Octets shall consist of exactly four or eight octets, respectively.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) The Contents Octets shall contain floating point values defined in conformance with ANSI/IEEE Standard 754. The sign is encoded by using bit 8 of the first octet. It is followed by the exponent starting from bit 7 of the first octet, and then the mantissa starting from bit 7 of the second octet for Floating32 and from bit 4 of the second octet for Floating64.

4.2.1.3.3.8 Encoding of a Variable-Length BitString value

- 1) The encoding of a variable-length BitString value shall be primitive.

- 2) The Identifier Octet shall not be present.
- 3) The Length Octet(s) shall indicate the number of octets in the BitString encoding, which is the number of octets used to encode a BitString value plus one for the "Number of Unused Bits" field (see (4)).
- 3.1) If the BitString is empty (no BitString value at all), there shall be no subsequent octets, and the Length Octet(s) shall be zero.
- 4) If the BitString is not empty, the Contents Octets shall encode the Number of Unused Bits and the BitString value as follows:
- 4.1) The Number of Unused Bits field shall follow the Length Octet(s) and shall always be one octet. It shall encode as an Unsigned8 type the number of unused bits in the final octet of the Contents Octets. This number shall be in the range zero to seven, inclusive.

NOTE1 The Number of Unused Bits field is provided for the service user.

NOTE2 The Number of Unused Bits is selected in accordance with ISO/IEC 8825.

- 4.2) The value of the BitString, commencing with the first bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first octet which follows the Number of Unused Bits field, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet up to and including the last octet of the Contents Octets. The unused bits shall be placed in bits 7 to 1 of the last octet, and their values are not specified.

4.2.1.3.3.9 Encoding of a Fixed-Length BitString Value

- 1) The encoding of a fixed-length Bitstring value of BitString8, BitString16, and BitString32 types shall be primitive, and the Contents Octets shall consist of exactly one, two, or four octets, respectively.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) A BitString value, commencing with the first bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet up to and including the last octet of the Contents Octets.

4.2.1.3.3.10 Encoding of a Variable-Length OctetString Value

- 1) The encoding of a variable-length OctetString value shall be primitive.
- 2) The Identifier Octet shall not be present.
- 3) The Length Octet(s) shall indicate as a binary number the number of octets in the OctetString value.
- 3.1) If the OctetString is of zero length, there shall be no subsequent octets, and the Length Octet(s) shall be zero.
- 4) If the OctetString is not of zero length, the Contents Octets shall be equal in value to the octets in the data value, in the order they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octets.

4.2.1.3.3.11 Encoding of a Fixed-Length Octet String Value

- 1) The encoding of a fixed-length OctetString value of OctetString2, OctetString4, OctetString8, and OctetString16 types shall be primitive, and the Contents Octets shall consist of exactly two, four, eight, or 16 octets, respectively.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) The Contents Octets shall be equal in value to the octets in the data value, in the order they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octets.

4.2.1.3.3.12 Encoding of a Variable-Length VisibleString Value

- 1) The encoding of a VisibleString value shall be primitive.
- 2) The Identifier Octet shall not be present.

- 3) The Length Octet(s) shall indicate as a binary number the number of octets in the VisibleString value.
- 3.1) If the VisibleString is of zero length, there shall be no subsequent octets, and the Length Octet(s) shall be zero.
- 4) If the VisibleString is not of zero length, the Contents Octets shall be equal in value to the octets in the data value, in the order they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octets.

4.2.1.3.3.13 Encoding of a Fixed-Length VisibleString Value

- 1) The encoding of a fixed-length VisibleString value of VisibleString2, VisibleString4, VisibleString8, and VisibleString16 types shall be primitive, and the Contents Octets shall consist of exactly two, four, eight, or 16 octets, respectively.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) The Contents Octets shall be equal in value to the octets in the data value, in the order they appear in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octets.

4.2.1.3.3.14 Encoding of an ISO10646String Value

- 1) The encoding of an ISO10646String value shall be primitive, and the Contents Octets shall consist of zero or an even number of octets.
- 2) The Identifier Octet shall not be present.
- 3) The Length Octet(s) shall indicate as a binary number the number of octets in the Contents Octets.
- 3.1) If the ISO10646String is of zero length, there shall be no subsequent octets, and the Length Octet(s) shall be zero.
- 4) If an ISO10646String value is not of zero length, the Contents Octets shall be equal in value to the octets in the data value.
- 4.1) Each ISO10646 character shall be placed in two octets in the Contents Octets, with the high order byte placed in the first octet and the low-order byte in the subsequent octet, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octets.

4.2.1.3.3.15 Encoding of a UniversalTime Value

- 1) The encoding of a UniversalTime value shall be primitive, and the Contents Octets shall consist of 12 octets.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) The Contents Octets shall be equal in value to the octets in the data value, and with the most significant bit of an octet of the data value aligned with the most significant bit of an octet of the Contents Octets.
- 3.1) The first "D" of a UniversalTime value shall be placed in the first octet of the Contents Octets, followed by the second "D" in the second octet, up to the second "s" in the 12th octet of the Contents Octets.
- 3.2) A UniversalTime value consists of two sets, each of six characters. The first set is YYMMDD, where YY is the two low-order digits of the Gregorian calendar, MM is the month (counting January as 01 and December as 12), and DD is the day of the month (01 to 31). The second set is hhmmss, where hh is hour (00 to 23), mm is minutes (00 to 59), and ss is seconds (00 to 59).
- 3.2) In countries where the Gregorian calendar is not adapted, it is possible to substitute the Gregorian YYMM with their local calendar convention. If a local calendar system is adopted, YY shall be the two low-order digits of that calendar system and MM shall be the month counting the first month of the calendar system as 01.

4.2.1.3.3.16 Encoding of Binary Time Value

- 1) The encoding of a BinaryTime0, BinaryTime1, BinaryTime2, BinaryTime3, BinaryTime4, BinaryTime5, BinaryTime6, BinaryTime7, BinaryTime8 and BinaryTime9 value shall be primitive.
- 2) The Identifier Octet and Length Octet(s) shall not be present.
- 3) The Contents Octets shall be a binary number equal to the binary time value, and consisting of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each octet in turn up to and including the last octet of the Contents Octets.
- 3.1) The Contents Octets of a BinaryTime0, BinaryTime1, BinaryTime2, and BinaryTime3 value shall consist of two octets.
- 3.2) The Contents Octets of a BinaryTime4, BinaryTime5, BinaryTime6, and BinaryTime7 value shall consist of four octets.
- 3.3) The Contents Octets of a BinaryTime8 and BinaryTime9 value shall consist of six octets.

NOTE The value of the granularity of each BinaryTime type is defined in IEC 61158-5.

4.2.1.3.3.17 Encoding of a CompactBCDArray Value

- 1) The encoding of a CompactBCDArray value shall be primitive.
- 2) The Identifier Octet shall not be present.
- 3) The Length Octet(s) shall indicate as a binary number the number of octets in the array.
- 3.1) If the number of BCD values is zero, there shall be no subsequent octets, and the Length Octet(s) shall be zero.
- 4) The Contents Octets shall encode the BCD values as the first BCD value shall be placed as a binary number in bits 8 to 5 of the first Contents Octets, the second BCD value shall be placed in bits 4 to 1 of the first Contents Octets. This will be repeated for the remaining BCD values and Contents Octets up to and including the last octet of the Contents Octets. The values of any unused bits in the last Contents Octets shall be set to F₁₆.

4.2.1.3.3.18 Encoding of an Array Value

- 1) An Array value shall be encoded as a SEQUENCE OF value.

4.2.1.3.3.19 Encoding of a Structure Value

- 1) A Structure value shall be encoded as a SEQUENCE value.

4.2.1.3.3.20 Encoding of the ANY Type

NOTE1 The ANY type is used to convey a value of an APO, such as a data value to be written to a variable object. Since an APO is not managed by the FAL, it does not provide any encoding means for this type of value. It is assumed that the service users know the data types of such data values.

- 1) The encoding of the ANY type shall be primitive or constructed.
- 2) The Length Octet(s) shall be present and shall indicate the number of octets in the Contents Octets.

NOTE2 This ensures that the first octet in the ANY type always indicates the total octets of the ANY type.

- 3) The Contents Octets shall be the encoding of a data value.
- 3.1) The data value shall be encoded by the service user so that the resultant encoding is identical to what is defined in this standard.

NOTE3 If a data to be encoded as ANY is of type Integer, for example, the service user is responsible for providing octets that represent the value of the Integer value based on the Integer encoding rule defined in this standard.

- 3.2) If the encoding of a data value is primitive, the Length Octet(s) of the data value, if it is present, shall be omitted.

- 3.3) If an Integer value is encoded as ANY, the Contents Octets of ANY include only the Contents Octets of the Integer value. Its Length Octet(s) shall be omitted.
- 3.4) If the encoding of a data value is constructed, the Length Octet(s) of the data value, if it is present, shall be present in the Contents Octets of ANY.

4.2.1.3.3.21 Encoding of a BCD Value

- 1) A BCD value shall be encoded as an Unsigned8 value.
- 2) A BCD value shall be placed in bits 4 to 1 of the Contents Octets of an Unsigned8 value. The values of the bits 8 to 5 shall be zero (0).

4.2.1.3.3.22 Encoding of a FieldbusTime Value

- 1) A FieldbusTime value shall be encoded as a BitString32 value.

4.2.1.3.3.23 Encoding of a Compact Boolean Array Value

- 1) A Compact Boolean Array value shall be encoded as a BitString value.

4.2.1.3.3.24 Key words Encoding Rules

The encoding rules for the following key words are defined:

- SEQUENCE
- SEQUENCE OF
- CHOICE
- TAGGEDTYPE
- IMPLICIT
- OPTIONAL
- DEFAULT

Encoding of a SEQUENCE value

- 1) The encoding of a SEQUENCE value shall be constructed.
- 2) The Identifier Octet, if it is present, shall have the class number of FAL -Specific 1.
 - 2.1) The Identifier Octet shall be omitted from the encoding of the SEQUENCE value used in the FAL syntax descriptions specified in clause 4.
- 3) The Length Octet(s), if it is present, shall indicate, as a binary number, the number of productions (not the number of octets) in the SEQUENCE value to be encoded.
 - 3.1) If there are zero data values, there shall be no subsequent octets, and the Length Octet(s) shall be zero.
 - 3.2) The Length Octet(s) shall be omitted from the encoding of the SEQUENCE value used in the FAL syntax descriptions specified in clause 4.

NOTE Rules 2.1) and 3.2), and the fact that all the APDU abstract syntax definitions are referenced with the IMPLICIT key word, ensure that the encoding of the APDU body starts with the encoding of the first component of each APDU productions, not with the encoding of SEQUENCE or SEQUENCE OF key words that envelop the productions.
- 4) The Contents Octets shall consist of the complete encoding of one data value from each of the types listed in the SEQUENCE type, in the order of their appearance in the definition, unless the type was referenced with the keyword "OPTIONAL" or the keyword "DEFAULT."

Encoding of a SEQUENCE OF Value

- 1) The encoding of a SEQUENCE OF value shall be constructed.
- 2) The Identifier Octet shall be present. The class number of the SEQUENCE OF value shall be FAL-Specific 1.
- 3) The Length Octet(s) shall indicate as a binary number the number of productions (not the number of octets) in the SEQUENCE OF value to be encoded.

- 3.1) If there are zero productions, there shall be no subsequent octets, and the Length Octet(s) shall be zero.
- 4) The Contents Octets shall consist of the complete encoding of the data values from the type listed in the SEQUENCE OF value. The order of the encoding of the data values shall be the same as the order of the data values in the SEQUENCE OF value to be encoded.

Encoding of Productions with OPTIONAL and DEFAULT Key Word

- 1) The encoding of a data value may, but need not, be present for a type which was referenced with the keyword "OPTIONAL" or the keyword "DEFAULT."
- 2) If a SEQUENCE contains productions with the OPTIONAL or DEFAULT key word, those productions shall not be tagged and, in the abstract syntax constructions to be used with this encoding rule, must be placed at the end of the SEQUENCE type. Then a production

optionalParametersMap Gn_OptionalParametersMap8

or

optionalParametersMap Gn_OptionalParametersMap16

or

optionalParametersMap Gn_OptionalParametersMap32

shall be placed just before the first OPTIONAL or DEFAULT production.

- 3) The "optionalParametersMap" parameter, of type Gn_OptionalParametersMap8, Gn_OptionalParametersMap16, or Gn_OptionalParametersMap32 shall indicate only the OPTIONAL or DEFAULT productions which are defined in the abstract syntax.
- 4) The "optionalParametersMap" parameter is one of the fixed-length BitString types, and the value of one in each of the bits indicates that the corresponding OPTIONAL or DEFAULT production is present in the encoding, and the value of zero indicates that it is not present.
- 5) The least significant bit of the optionalParametersMap parameter corresponds to the first OPTIONAL or DEFAULT production in the SEQUENCE, the second least significant bit corresponds to the second production, and the n-th bit corresponds to the n-th production.
- 6) If a SEQUENCE value contains more than 32 OPTIONAL or DEFAULT productions, the BitString type shall be used as the type of the optionalParametersMap parameter.
- 6.1) The Number of Unused Bits field of this BitString value shall be zero since the bits are used beginning from the lsb as described above.
- 7) If it is possible that all the optional parameters are not present in an encoding, the optionalParametersMap parameter itself may become OPTIONAL.
- 7.1) When the optionalParametersMap parameter is OPTIONAL, there shall be no additional optionalParametersMap parameters present to indicate that the optionalParametersMap parameter is OPTIONAL.

Encoding of a CHOICE Value

- 1) The encoding of a CHOICE value shall be the same as the encoding of a value of the chosen type.

NOTE In the abstract syntax constructions to be used with this encoding rule, each type listed as a CHOICE must be an unambiguously tagged type, since the Identifier Octet is frequently not encoded for the type specified.

Encoding of a Tagged Value

- 1) The encoding of a Tagged value shall be primitive if the referenced encoding is primitive and constructed if it is constructed.
- 2) The Identifier Octet shall contain the value of the tag and its class.
- 2.1) The Identifier Octet for the tagged value shall not be encoded unless those used with the CHOICE type.
- 3) The Length Octet(s) may or may not be present.

NOTE These rules ensure that the Length Octet(s) of the Tagged value always indicates the number of octets of the Contents Octet(s) of the Tagged value.

- 3.1) If the referenced encoding is primitive and does not have the Length Octet(s), the Length Octet(s) of the Tagged value shall indicate the number of octets of the Contents Octet(s) of the referenced encoding.
- 3.2) If the referenced encoding is primitive and has the Length Octet(s), the Length Octet(s) of the referenced encoding shall be placed in the Length Octet(s) of the Tagged value.
- 3.3) If the referenced encoding is constructed, the Length Octet(s) of the Tagged value shall indicate the number of components of the referenced encoding.
- 4) The Contents Octet(s) shall be those of the referenced encoding.

Encoding of an IMPLICIT Value

- 1) The IMPLICIT keyword shall only be used with a Tagged type value, or one of the APDU class specifiers defined in the "Encoding of Type Field" clause.
- 2) If the IMPLICIT keyword is used, the Identifier Octet of the type which immediately follows the keyword shall not be present.

NOTE If a base encoding does not have the Identifier Octet and/or Length Octet(s), the IMPLICIT keyword does not affect the encoding.

4.2.1.3.4 Buffer-Oriented Encoding Rules (BER) and Messaging Encoding Rule (MER)

4.2.1.3.4.1 Encoding Rule for Buffer Services (BER)

Introduction

The Encoding Rule for Buffer Services (BER) is an encoding rule for new implementation where Time critical transfer is selected.

Application Layer Encoding Rules

Overview of Encoding

The PDUs that conform to this standard shall be encoded in a uniform format as shown in the figure below. The PDUs consist of two major parts: the "APDU Header" part and the "APDU Body" part.

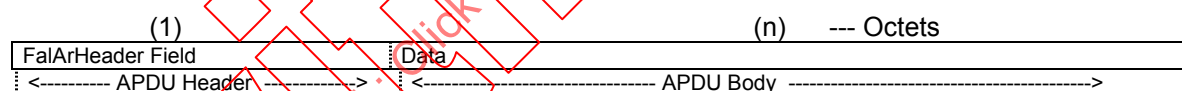


Figure 22 – APDU Overview

APDU Header Encoding

The APDU Header part is always present in all APDUs which conform to this part of IEC 61158-2. It consists of one field: the FalArHeader Field. Refer to 4.2.1.1 for the encoding rule of the FalArHeader field.

APDU Body Encoding

The FAL encoding rules are based on terms and conventions defined in ISO/IEC standards. The encoding consists of three components in the following order:

- An Identifier Octet
- Length Octet(s)
- The ContentsOctets

Identifier Octet

- 1) The Identifier Octet shall encode the type and shall consist of one octet.
- 2) The value of this octet is 40 h. Other values are reserved for management.

Length Octet

- 1) The Length Octet shall consist of one octet.

ContentsOctets

1) The ContentsOctets shall encode the data value as a string of octet.

4.2.1.3.4.2 Encoding Rule for Messaging Services (MER)

The encoding of an ASN.1 type value may comprise the following components:

- Identification Octet
- Content Length Octets
- Content Octets

Identification Octet

The Identification Octet is used for the encoding of the type tag associated with the value. It contains the class and number of the type tag.

The Identification Octet only exists in the encoding of tagged types appearing in a CHOICE. If present, the Identification Octet is structured as follows:

- bit 8 defines the value type class:

Table 13 – Identification Octet Classes

class	bit 8 value
FAL specific	0
Context-specific	1

- bits 7 to 1 are used for encoding the identifier numbers in the range 0 to 127, in the form of an unsigned binary number, whose most and least significant bits are respectively bits 7 and 1.

Contents length octets

If present, the encoding of the contents length of the type value always consists of two octets representing a number of units of measurement (unsigned binary number).

The unit of measurement is type-dependent, as follows:

Table 14 – Unit of Measurement of Contents Length Octets

SEQUENCE OF	octet
SEQUENCE	octet
INTEGER	octet
BOOLEAN	octet
BIT STRING	bit
OCTET STRING	octet

ContentsOctets

The ContentsOctets shall encode the data value according to the encoding rules defined for the respective type as specified in the following subclauses. The ContentsOctets may consist of zero, one or more octets.

4.2.1.3.4.3 Type Encoding Rules**Boolean**

A Boolean value shall be encoded as follows:

- The Identifier Octet and the Length Octets shall not be present.
- The ContentsOctets always consist of one byte. If the Boolean value equals FALSE, all bits of the octet are 0. If the Boolean value equals TRUE, the octet can contain any combination of bits other than the encoding for FALSE.

Integer

An integer value shall be encoded as follows:

- The Identifier Octet shall not be present.

- The Length Octets shall not be present, if the size of the integer type is invariable. An integer with invariable size is created by constraining the possible value. The length octets shall be present, if the size of the integer value is variable.

The ContentsOctets shall contain the two complement binary number equal to the integer value. The most significant bits of the integer value are encoded in bit8 to bit1 of the first octet, the next in bit8 to bit1 of the next octet and so on. If the values of an integer type are restricted to negative and non-negative numbers, bit8 of the first octet gives the sign of the value, if the values are restricted to non-negative numbers only, no sign bit is needed (see examples below)

Examples:

- INTEGER (-128..128) is an integer with invariable octet length of one.
- INTEGER (0..255) is also an integer with invariable octet length of one.
- INTEGER is an integer with variable octet length.

Bit string

A bitstring value shall be encoded as follows:

- The Identifier Octet shall not be present
- The Length Octets shall not be present, if the size of the bitstring type is invariable. A bitstring with invariable size is created by applying a size constraint containing only one value on the bitstring type. The length octets shall be present, if the size of the bitstring value is variable.
- The ContentsOctets comprise as many octets as necessary to contain all bits of the actual value: $N\_Octets = (N\_Bits - 1) \div 8 + 1$. The bitstring value commencing with the first bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet and so on. If the number of bits is not a multiple of 8 there are so-called unused bits, which are located in the least significant bits of the last octet. The value of the unused bits may be zero or one and carry no meaning.

Examples:

- BIT STRING SIZE (30) is a bitstring with invariable octet length of four and two unused bits.
- BIT STRING SIZE (1..32) is a bitstring with variable octet length.
- BIT STRING is a bitstring with variable octet length.

Octet string

An octetstring value shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall not be present, if the size of the octetstring type is invariable. An octetstring with invariable size is created by applying a size constraint containing only one value on the octetstring type. The length octets shall be present, if the size of the octetstring value is variable.
- The ContentsOctets shall be equal in value to the octets in the data value.

Examples:

- OCTET STRING SIZE (30) is an octetstring with invariable octet length of 30.
- OCTET STRING SIZE (1..32) is an octetstring with variable octet length.
- OCTET STRING is an octetstring with variable octet length.

Visible string

A visiblestring value shall be encoded as follows:

- The Identifier Octet shall not be present.

- The Length Octets shall not be present, if the size of the visiblestring type is invariable. A visiblestring with invariable size is created by applying a size constraint containing only one value on the visiblestring type. The length octets shall be present, if the size of the visiblestring value is variable.
- The ContentsOctets shall be equal in value to the octets in the data value.

Examples:

- VisibleString SIZE (30) is a visiblestring with invariable octet length of 30.
- VisibleString SIZE (1..32) is a visiblestring with variable octet length.
- VisibleString is a visiblestring with variable octet length.

SEQUENCE Types

A value of a SEQUENCE type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall be there and specify the number of octets being used for the ContentsOctets. However, for the first Keyword "SEQUENCE" of FaArPDU, this length shall not be encoded at all.
- The ContentsOctets shall consist of the encoding of all the element types in the same order as they are specified in the ASN.1 description of the SEQUENCE type.

SEQUENCE OF Types

A value of a SEQUENCE OF type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall be there and specify the number of octets being used for the ContentsOctets.
- The ContentsOctets shall consist of the encoding of all the element types in the same order as they are specified in the ASN.1 description of the SEQUENCE OF type.

CHOICE Types

A value of a CHOICE type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall not be present.
- The ContentsOctets shall consist of the encoding of the selected type of the alternative type list.

Null

A value of a NULL type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall not be present.
- The ContentsOctets shall not be present.

Tagged Types

A value of a tagged type shall be encoded as follows:

- The Identifier Octet shall only be present, if the tagged type is part of an Alternative Type List in a CHOICE construct.
- The Length Octets shall not be present.
- The ContentsOctets shall consist of the encoding of the Type which was tagged.

IMPLICIT Types

A value of an IMPLICIT type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall not be present.
- The ContentsOctets shall consist of the encoding of the Type being referenced by the IMPLICIT construct, except for the case when the referenced Type is a SEQUENCE type. In this case, the ContentsOctets consist only of the ContentsOctets of the referenced SEQUENCE type, the length information of this SEQUENCE type should not occur in the encoding.

OPTIONAL and DEFAULT Types

For the encoding of a value of an OPTIONAL or DEFAULT type there are two possibilities:

1) If the OPTIONAL/DEFAULT type is a subtype of a SEQUENCE type containing a Gn_OptionalParametersMap element, which is used to determine the existence or non-existence of all optional subtypes within this structure. The least significant bit of this Gn8OptionalParametersMap type corresponds to the first OPTIONAL or DEFAULT subtype, the second least significant bit corresponds to the second subtype and so on.

In this case, a value of an OPTIONAL or DEFAULT type shall be encoded as follows:

If the corresponding bit of the Gn_OptionalParametersMap is zero or the Gn_OptionalParametersMap is missing altogether (it may be an OPTIONAL itself), the OPTIONAL/DEFAULT value does not exist and should not be encoded at all. If the corresponding bit of the Gn_OptionalParametersMap is one, the value exists, and its encoding should be as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall not be present.
- The ContentsOctets shall consist of the encoding of the Type being referenced by the OPTIONAL/DEFAULT construct.

2) Otherwise, a value of an OPTIONAL or DEFAULT type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall be present. If there is no value for this type, the Length Octets contain the value 0.
- The ContentsOctets shall consist of the encoding of the referenced Type, if there is a value for this type, otherwise no ContentsOctets exist.

If a Gn_OptionalParametersMap type is defined as OPTIONAL itself, it is always encoded in the second way.

ANY Types

An ANY type is used for the definition of complex types, whose structure is described informally rather than in ASN.1, generally because its definition in ASN.1 would not result in an optimal encoding. Usually the structuring information (Identifier Octets, Length Octets) in the encoding of such types can be omitted, since the receiver of the encoded type has additional knowledge, which enables him to encode such an optimized PDU.

A value of an ANY type shall be encoded as follows:

- The Identifier Octet shall not be present.
- The Length Octets shall not be present.
- The ContentsOctets shall consist of the encoding of all implicit types which constitute the ANY type.

Encoding of APDU Header

The APDU Header always begins with the FALArHeader. This implies that the Contents Length of the surrounding SEQUENCE is not encoded.

4.3 FAL Protocol State Machines Structure

Interface to FAL services and protocol machines are specified in this section.

NOTE The state machines specified in this section and ARPMs defined in the following sections only define the valid events for each. It is a local matter to handle these invalid events.

The behavior of the FAL is described by three integrated protocol machines. Specific sets of these protocol machines are defined for different AREP types. The three protocol machines are: FAL Service Protocol Machine (FSPM), the Application Relationship Protocol Machine (ARPM), and the Data Link Layer Mapping Protocol Machine (DMPM). The relationship among these protocol machines as well as primitives exchanged among them are depicted in the following figure:

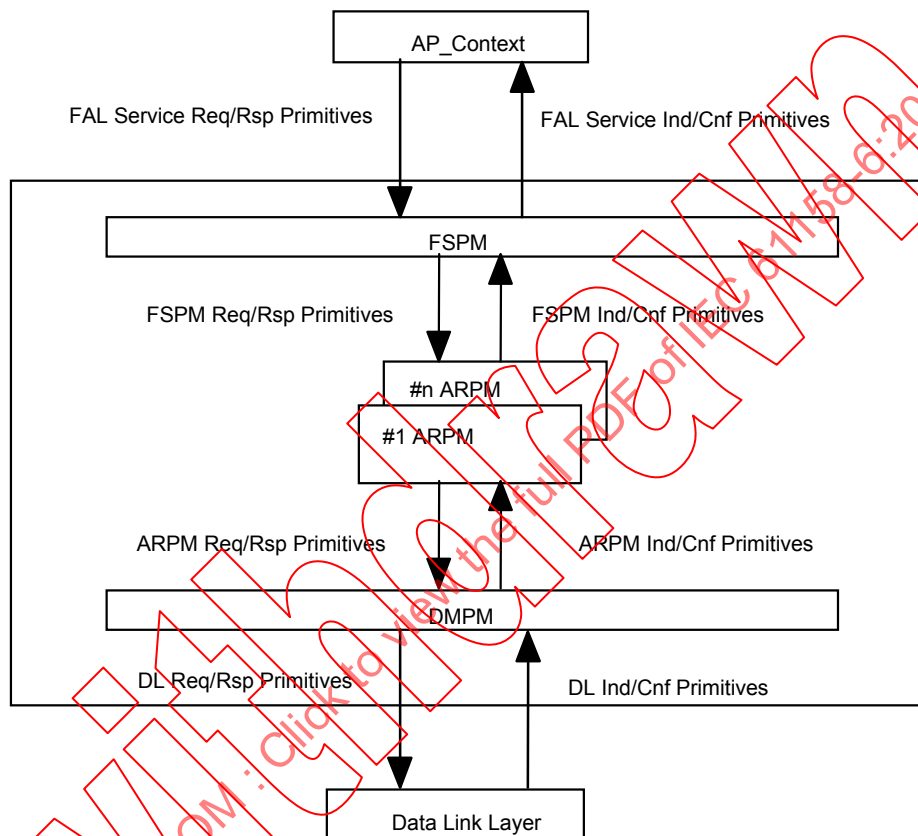


Figure 23 – Relationships among Protocol Machines and Adjacent Layers

- The FSPM describes the service interface between the AP_Context and a particular AREP. The FSPM is common to all the AREP classes and does not have any state changes. The FSPM is responsible for the following activities:
 - a) to accept service primitives from the FAL service user and convert them into FAL internal primitives;
 - b) to select an appropriate ARPM state machine based on the AREP Identifier parameter supplied by the AP_Context and send FAL internal primitives to the selected ARPM;
 - c) to accept FAL internal primitives from the ARPM and convert them into service primitives for the AP_Context;
 - d) to deliver the FAL service primitives to the AP_Context based on the AREP Identifier parameter associated with the primitives.
- The ARPM describes the establishment and release of an AR and exchange of FAL-PDUs with a remote ARPM(s). The ARPM is responsible for the following activities:
 - a) to accept FAL internal primitives from the FSPM and create and send other FAL internal primitives to either the FSPM or the DMPM, based on the AREP and primitive types;

- b) to accept FAL internal primitives from the DMPM and send them to the FSPM as a form of FAL internal primitives;
- c) if the primitives are for the Establish or Abort service, it shall try to establish or release the specified AR.
- The DMPM describes the mapping between the FAL and the DLL. It is common to all the AREP types and does not have any state changes. The DMPM is responsible for the following activities:
 - a) to accept FAL internal primitives from the ARPM, prepare DLL service primitives, and send them to the DLL;
 - b) to receive DLL indication or confirmation primitives from the DLL and send them to the ARPM in a form of FAL internal primitives.

4.4 AP Context State Machine

4.4.1 Primitive Definitions

4.4.1.1 Primitives Exchanged between FAL-User and AP-Context

Table 15 – Primitives issued by FAL-User to AP-Context

Primitive Name	Source	Associated Parameters and Functions
Terminate.req	FAL-User	Refer to FAL Service Definition (IEC 61158-5)
Initiate.req	FAL-User	
Initiate.rsp(+)	FAL-User	
Initiate.rsp(-)	FAL-User	
ConfirmedService.req	FAL-User	
ConfirmedService.rsp	FAL-User	
UnconfirmedService.req	FAL-User	

Table 16 – Primitives issued by AP-Context to FAL-User

Primitive Name	Source	Associated Parameters and Functions
Terminate.ind	AP-Context	Refer to FAL Service Definition (IEC 61158-5)
Initiate.ind	AP-Context	
Initiate.cnf(+)	AP-Context	
Initiate.cnf(-)	AP-Context	
ConfirmedService.ind	AP-Context	
ConfirmedService.cnf	AP-Context	
UnconfirmedService.ind	AP-Context	

4.4.2 State Machine Description

The attributes used in this state machine are defined as elements of the AP attribute *List Of In-Use AR Endpoint Info*.

CLOSED

The AP Context for an AR is not open. Only the service primitives Initiate.req, Establish.ind, Terminate.req and Abort.ind are allowed. All other service primitives shall be rejected with the Terminate service.

OPENING-REQUESTING (REQ)

The local FAL user wishes to open the AP Context for an AR. Only the service primitives Establish.cnf(+), Establish.cnf(-), Terminate.req and Abort.ind are allowed. All other services shall be rejected with the Terminate service.

OPENING-RESPONDING (RSP)

The remote AP Context wishes to open the AP Context for an AR. Only the service primitives Initiate.rsp(+), Initiate.rsp(-), Terminate.req and Abort.ind are allowed. All other service primitives shall be rejected with the Terminate service.

OPEN

The AP Context for an AR is open. The service primitives Initiate.req, Initiate.rsp(+), Initiate.rsp(-) are not allowed and shall be rejected with the Terminate service. The following actions shall be taken to reset the AP Context for an AR.

- Set the *Outstanding Services Requesting Counter* and *Outstanding Services Responding Counter* to 0.
- Set the *Context State* to "CLOSED"

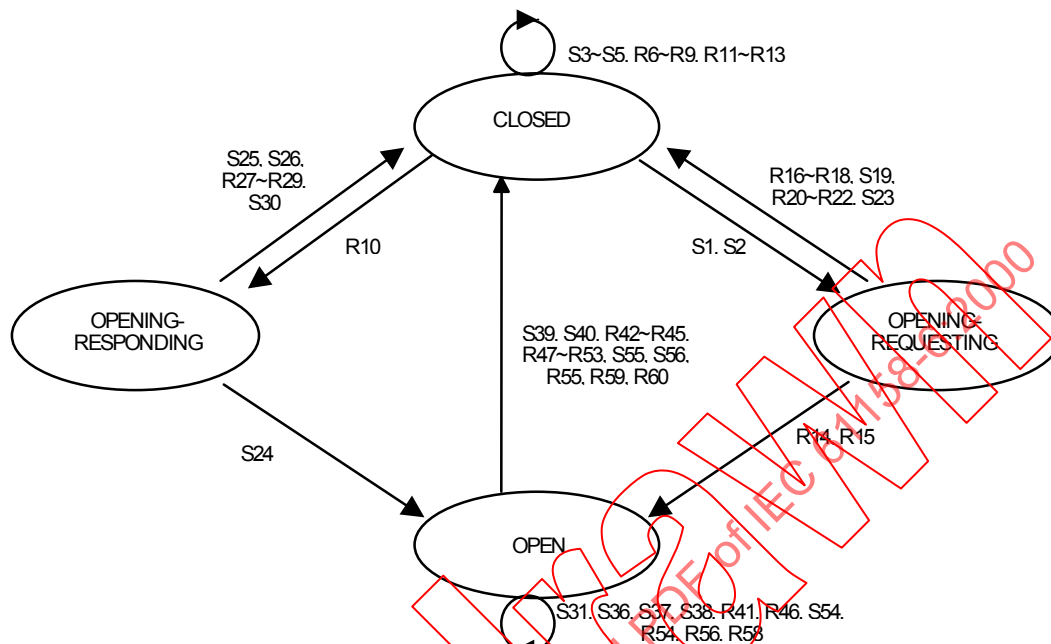


Figure 24 – AP-AP Context Initiation State Machine

4.4.3 AP-AP Context Initiation State Transitions

In the tables of this subclause "Data" parameter for FAL Service Protocol Machine is expressed by "FalApduBody" to emphasize its semantics.

NOTE Since Type 1 defines different abstract syntax for the same service, multiple PDU forms are used in the following state tables. It is assumed and advised that only one selection that uses the same abstract syntax will be used in any instances of these state machines.

Table 17 – AP Context State Machine Sender Transactions

#	Current State	Events Actions	Next State
S1	CLOSED	Initiate.req && ApContextTest = "True" && ApExplicitConnection = "True" EST.req { FalApduBody := "Establish-CommandPDU" } or EST.req { FalApduBody := "Establish-RequestPDU" } or EST.req { FalApduBody := "Establish-Request2PDU" }	REQ
S2	CLOSED	Initiate.req && ApContextTest = "True" && ApExplicitConnection = "False" EST.req { FalApduBody := "NULL" }	REQ

#	Current State	Events Actions	Next State
S3	CLOSED	Initiate.req && ApContextTest = ""False"" Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "Context error" }	CLOSED
S4	CLOSED	Terminate.req (no actions taken)	CLOSED
S5	CLOSED	FAL service.primitive <> "Initiate" && FAL service.primitive <> "Terminate" Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "User error" }	CLOSED
S19	REQ	Terminate.req Abort.req { Originator := "Terminateldentifier of Terminate.req", ReasonCode := "ReasonCode of Terminate.req" }, ResetArep	CLOSED
S23	REQ	FAL service.primitive <> "Initiate.rsp" && FAL service.primitive <> "Terminate.req" Abort.req { Originator := "FAL", ReasonCode := "User error" }, Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "User error" }, ResetArep	CLOSED
S24	RSP	Initiate.rsp(+) EST.rsp(+){ FalAduBody := "Establish-AffirmativePDU" } or EST.rsp(+){ FalAduBody := "Establsih-ResponsePDU" }	OPEN
S25	RSP	Initiate.rsp(-) EST.rsp(-){ FalAduBody := "Establish-NegativePDU" }, or EST.rsp(-){ FalAduBody := "Establsih-ErrorPDU" }, ResetArep	CLOSED
S26	RSP	Terminate.req Abort.req { Originator := "Terminateldentifier of Terminate.req", ReasonCode := "ReasonCode of Terminate.req" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
S30	RSP	FAL service.primitive <> "Initiate.rsp" && FAL service.primitive <> "Terminate.req" Abort.req { Originator := "FAL", ReasonCode := "User error" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "User error" }, ResetArep	CLOSED
S31	OPEN	ConfirmedService.req && ConfirmedServiceCheck = "True" && OutstandingServicesRequestingCounter < NegotiatedMaxOutstanding-ServicesRequesting && InvokeldExistent = "False" && PDU length ≤ NegotiatedMaxPduSizeSending && RequestedServiceSupportedTest = "True" CS.req { FalApduBody := "ConfirmedSend-CommandPDU" }, or CS.req { FalApduBody := "ConfirmedSend-RequestPDU" }, OutstandingServicesRequestingCounter := OutstandingServicesRequestingCounter+1	OPEN
S36	OPEN	UnconfirmedService.req && UnconfirmedServiceCheck = "True" && PDU length ≤ NegotiatedMaxPduSizeSending && RequestedServiceSupportedTest = "True" && ImmediateAcknowledge = "False" UCS.req { FalApduBody := "UnconfirmedSend-CommandPDU" } or UCS.req { FalApduBody := "UnconfirmedSend-PDU" }	OPEN
S37	OPEN	UnconfirmedService.req && UnconfirmedServiceCheck = "True" && PDU length ≤ NegotiatedMaxPduSizeSending && RequestedServiceSupportedTest = "True" && ImmediateAcknowledge = "True" UCA.req { FalApduBody := "UnconfirmedAcknowledgedSend-CommandPDU" } or UCA.req { FalApduBody := "UnconfirmedAcknowledgedSend-RequestPDU" }	OPEN
S38	OPEN	AR_ASE service request && ArServiceCheck = "True" && RequestedServiceSupportedTest = "True" ArFspmService NOTE This state is provided to access the FSPM direct from the FAL User. The function ArFspmService generates an FSPM primitive as defined later in this section.	OPEN
S39	OPEN	Terminate.req Abort.req { Originator := "TerminatIdentifier of Terminate.req", ReasonCode := "ReasonCode of Terminate.req" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
S40	OPEN	Faulty, unknown or not-allowed FAL service.primitive <pre> Abort.req { AbortIdentifier := "FAL", ReasonCode := "User error" }, Terminate.ind { LocallyGenerated := "True", TerminateIdentifier := "FAL", ReasonCode := "User error" }, ResetArep </pre>	CLOSED
S54	OPEN	<pre> ConfirmedService.rsp && ConfirmedServiceCheck = "True" && InvokeIdExistent = "True" && SameService = "True" && PDU length ≤ NegotiatedMaxPduSizeSending CS.rsp { FalApuBody := "ConfirmedSend-AffirmativePDU" }, or CS.rsp { FalApuBody := "ConfirmedSend-NegativePDU" }, or CS.rsp { FalApuBody := "ConfirmedSend-ResponsePDU" }. OutstandingServicesRespondingCounter := OutstandingServicesRespondingCounter-1 NOTE An extra protocol means shall be provided to determine whether ConfirmedSend- AffirmativePDU or ConfirmedSend-NegativePDU is used. </pre>	OPEN
S55	OPEN	<pre> ConfirmedService.rsp && ConfirmedServiceCheck = "True" && InvokeIdExistent = "False" Abort.req { Originator := "FAL", ReasonCode := "InvokeId-error-response" }, Terminate.ind { LocallyGenerated := "True", TerminateIdentifier := "FAL", ReasonCode := "InvokeId-error-response" }, ResetArep </pre>	CLOSED
S56	OPEN	<pre> ConfirmedService.rsp && ConfirmedServiceCheck = "True" && InvokeIdExistent = "True" && SameService = "False" Abort.req { Originator := "FAL", ReasonCode := "Service-error" }, Terminate.ind { LocallyGenerated := "True", TerminateIdentifier := "FAL", ReasonCode := "Service-error" }, ResetArep </pre>	CLOSED

Table 18 – AP Context State Machine Receiver Transactions

#	Current State	Events Actions	Next State
R6	CLOSED	Abort.ind (no actions taken)	CLOSED
R7	CLOSED	Faulty or unknown AR_FSPM service primitive Abort.req { Originator := "FAL", ReasonCode := "AR_ASE error" }	CLOSED
R8	CLOSED	AR_FSPM service primitive <> "EST.ind" && AR_FSPM service.primitive <> "Abort.ind" Abort.req { Originator := "FAL", ReasonCode := "Connection State Conflict" }	CLOSED
R9	CLOSED	EST.ind && FalApduBody = not allowed, unknown or faulty FAL PDU Abort.req { Originator := "FAL", ReasonCode := "FAL PDU error" }	CLOSED
R10	CLOSED	EST.ind && (FalApduBody = "Establish-CommandPDU" FalApduBody = "Establish-RequestPDU" FalApduBody = "Establish-Request2PDU") && ApContextTest = "True" && MaxFalPduLengthTest = "True" && ServicesSupportedTest = "True" NegotiateOutstandingServices. Initiate.ind { }	RSP
R11	CLOSED	EST.ind && (FalApduBody = "Establish-CommandPDU" FalApduBody = "Establish-RequestPDU" FalApduBody = "Establish-Request2PDU") && ApContextTest = "False" Abort.req { Originator := "FAL", ReasonCode := "AR error" }	CLOSED
R12	CLOSED	EST.ind && (FalApduBody = "Establish-CommandPDU" FalApduBody = "Establish-RequestPDU" FalApduBody = "Establish-Request2PDU") && ApContextTest = "True" && MaxFalPduLengthTest = "False" EST.rsp(-) { FalApduBody := "Establish-NegativePDU", ErrorCode := "Max-Fal-Fdu-Size-Insufficient" } or EST.rsp(-) { FalApduBody := "Establish-ErrorPDU", ErrorCode := "Max-Fal-Fdu-Size-Insufficient" }	CLOSED

#	Current State	Events Actions	Next State
R13	CLOSED	EST.ind && (FalApduBody = "Establish-CommandPDU" FalApduBody = "Establish-RequestPDU" FalApduBody = "Establish-Request2PDU") && ApContextTest = "True" && MaxFalPduLengthTest = "True" && ServicesSupportedTest = "False" EST.rsp(-) { FalApduBody := "Establish-NegativePDU", ErrorCode := "Service-Not-Supported" } or EST.rsp(-) { FalApduBody := "Establish-ErrorPDU", ErrorCode := "Service-Not-Supported" }	CLOSED
R14	REQ	EST.cnf(+) && (FalApduBody = "Establish-AffirmativePDU" FalApduBody = "Establish-ResponsePDU") && ApExplicitConnection = "True" Initiate.cnf(+) {	OPEN
R15	REQ	EST.cnf(+) && FalApduBody = "NULL" && ApExplicitConnection = "False" Initiate.cnf(+) {	OPEN
R16	REQ	EST.cnf(-) && (FalApduBody = "Establish-NegativePDU" FalApduBody = "Establish-ErrorPDU") && ApExplicitConnection = "True" Initiate.cnf(-) { ErrorCode := "Errorcode of EST.cnf(-)" }, ResetArep	CLOSED
R17	REQ	EST.cnf(-) && FalApduBody = "NULL" && ApExplicitConnection = "False" Initiate.cnf(-) { ErrorCode := "Errorcode in EST.cnf" }, ResetArep	CLOSED
R18	REQ	Abort.ind Terminate.ind { LocallyGenerated := "LocallyGenerated of Abort.ind", TerminatIdentifier := "Originator of Abort.ind", ReasonCode := "ReasonCode of Abort.ind" }, ResetArep	CLOSED
R20	REQ	Faulty or unknown AR_FSPM service primitive Abort.req { Originator := "FAL", ReasonCode := "AR_ASE error" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "AR_ASE error" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
R21	REQ	AR_FSPM service.primitive <> "EST.cnf" && AR_FSPM service.primitive <> "Abort.ind" Abort.req { Originator := "FAL", ReasonCode := "Connection State Conflict" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "Connection State Conflict" }, ResetArep	CLOSED
R22	REQ	EST.cnf && FalApduBody = " not allowed, unknown or faulty FAL PDU" Abort.req { AbortIdentifier := "FAL", ReasonCode := "FAL PDU error" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "FAL PDU error" }, ResetArep	CLOSED
R27	RSP	Abort.ind Terminate.ind { LocallyGenerated := "LocallyGenerated of Abort.ind", TerminatIdentifier := "Originator of Abort.ind", ReasonCode := "ReasonCode of Abort.ind" }, ResetArep	CLOSED
R28	RSP	Faulty or unknown AR_FSPM service primitive Abort.req { AbortIdentifier := "FAL", ReasonCode := "AR_ASE error" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "AR_ASE error" }, ResetArep	CLOSED
R29	RSP	AR_FSPM service.primitive <> " Abort.ind" Abort.req { Originator := "FAL", ReasonCode := "State Conflict with AR_ASE" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "AP_ASE", ReasonCode := "Connection State Conflict with AR_ASE" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
R41	OPEN	CS.ind && (FalApduBody = "ConfirmedSend-CommandPDU" FalApduBody = "ConfirmedSend-RequestPDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && OutstandingServicesRespondingCounter < NegotiatedMaxOutstanding- ServicesResponding && InvokeldExistent = "False" && IndicatedServiceSupportedTest = "True" ConfirmedService.ind { }, OutstandingServicesRespondingCounter := OutstandingServicesRespondingCounter+1	OPEN
R42	OPEN	CS.ind && (FalApduBody = "ConfirmedSend-CommandPDU" FalApduBody = "ConfirmedSend-RequestPDU") && PDU length > NegotiatedMaxPduSizeReceiving Abort.req { Originator := "FAL", ReasonCode := "PDU-size" }, Terminate.ind { LocallyGenerated := "True", TerminateIdentifier := "FAL", ReasonCode := "PDU-size" }, ResetArep	CLOSED
R43	OPEN	CS.ind && (FalApduBody = "ConfirmedSend-CommandPDU" FalApduBody = "ConfirmedSend-RequestPDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && OutstandingServicesResponding ≥ NegotiatedMaxOutstanding-ServicesResponding Abort.req { Originator := "FAL", ReasonCode := "Max-services-overflow" }, Terminate.ind { LocallyGenerated := "True", TerminateIdentifier := "FAL", ReasonCode := "Max-Services-Overflow" }, ResetArep	CLOSED
R44	OPEN	CS.ind && (FalApduBody = "ConfirmedSend-CommandPDU" FalApduBody = "ConfirmedSend-RequestPDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && OutstandingServicesRespondingCounter < NegotiatedMaxOutstanding- ServicesResponding && InvokeldExistent = "True" Abort.req { Originator := "FAL", ReasonCode := "Invokeld-error-request" }, Terminate.ind { LocallyGenerated := "True", TerminateIdentifier := "FAL", ReasonCode := "Invokeld-error-request" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
R45	OPEN	CS.ind && (FalApuBody = "ConfirmedSend-CommandPDU" FalApuBody = "ConfirmedSend-RequestPDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && OutstandingServicesRespondingCounter < NegotiatedMaxOutstandingServicesResponding && InvokeldExistent = "False" && IndicatedServiceSupportedTest = "False" Abort.req { Originator := "FAL", ReasonCode := "Feature-not-supported" }, Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "Feature-not-supported" }, ResetArep	CLOSED
R46	OPEN	UCS.ind && (FalApuBody = "UnconfirmedSend-CommandPDU" FalApuBody = "UnconfirmedSend-PDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && IndicatedServiceSupportedTest = "True" UnconfirmedService.ind { }	OPEN
R47	OPEN	UCS.ind && (FalApuBody = "UnconfirmedSend-CommandPDU" FalApuBody = "UnconfirmedSend-PDU") && PDU length > NegotiatedMaxPduSizeReceiving Abort.req { Originator := "FAL", ReasonCode := "PDU-size" }, Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "PDU-size" }, ResetArep	CLOSED
R48	OPEN	UCS.ind && (FalApuBody = "UnconfirmedSend-CommandPDU" FalApuBody = "UnconfirmedSend-PDU") && PDU length > NegotiatedMaxPduSizeReceiving && IndicatedServiceSupportedTest = "False" Abort.req { Originator := "FAL", ReasonCode := "Feature-not-supported" }, Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "Feature-not-supported" }, ResetArep	CLOSED
R49	OPEN	Abort.ind Terminate.ind { LocallyGenerated := "LocallyGenerated of Abort.ind", Terminateldentifier := "Originator of Abort.ind", ReasonCode := "ReasonCode of Abort.ind" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
R50	OPEN	EST.ind && (FalAduBody = "Establish-CommandPDU" FalAduBody = "Establish-RequestPDU" FalAduBody = "Establish-Request2PDU") Abort.req { Originator := "FAL", ReasonCode := "Connection-State-Conflict" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "Connection-State-Conflict" }, ResetArep	CLOSED
R51	OPEN	Faulty or unknown AR_FSPM service primitive Abort.req { Originator := "FAL", ReasonCode := "AR_ASE error" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "AR_ASE error" }, ResetArep	CLOSED
R52	OPEN	Not-allowed AR_FSPM service primitive Abort.req { Originator := "FAL", ReasonCode := "Connection-State-Conflict" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "Connection-State-Conflict" }, ResetArep	CLOSED
R53	OPEN	valid AR_FSPM Send Service primitive (one of CS, US, UCA) && FalAduBody = not-allowed, unknown or faulty FAL PDU && ArAccessSupported = "False" Abort.req { Originator := "AP_ASE", ReasonCode := "FAL-PDU-error" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "FAL -PDU-error" }, ResetArep	CLOSED
R54	OPEN	UCA.ind && ArAccessSupported = "True" && (FalAduBody = "UnconfirmedAcknowledgedSend-CommandPDU" FalAduBody = "UnconfirmedAcknowledgedSend-RequestPDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving UnconfirmedService.ind { }	OPEN

#	Current State	Events Actions	Next State
R55	OPEN	UCA.ind && ArAccessSupported = "True" && (FalAduBody = "UnconfirmedAcknowledgedSend-CommandPDU" FalAduBody = "UnconfirmedAcknowledgedSend-RequestPDU") && PDU length > NegotiatedMaxPduSizeReceiving Abort.req { Originator := "FAL", ReasonCode := "PDU-size" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "PDU-size" }, ResetArep	CLOSED
R56	OPEN	AR_ASE service indication ArFspmService NOTE This state is provided to access the FSPM direct from the FAL User. The function ArFspmService generates an FSPM primitive as defined later in this section.	OPEN
R58	OPEN	CS.cnf && (FalAduBody = "ConfirmedSend-AffirmativePDU" FalAduBody = "ConfirmedSend-NegativePDU" FalAduBody = "ConfirmedSend-ResponsePDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && InvokeldExistent = "True" && SameService = "True" ConfirmedService.cnf { }, OutstandingServicesRequestingCounter := OutstandingServicesRequestingCounter-1	OPEN
R59	OPEN	CS.cnf && (FalAduBody = "ConfirmedSend-AffirmativePDU" FalAduBody = "ConfirmedSend-NegativePDU" FalAduBody = "ConfirmedSend-ResponsePDU") && PDU length > NegotiatedMaxPduSizeReceiving Abort.req { AbortIdentifier := "FAL", ReasonCode := "PDU-size" }, Terminate.ind { LocallyGenerated := "True", TerminatIdentifier := "FAL", ReasonCode := "PDU-size" }, ResetArep	CLOSED

#	Current State	Events Actions	Next State
R60	OPEN	CS.cnf && (FalApduBody = "ConfirmedSend-AffirmativePDU" FalApduBody = "ConfirmedSend-NegativePDU" FalApduBody = "ConfirmedSend-ResponsePDU") && PDU length ≤ NegotiatedMaxPduSizeReceiving && InvokeldExistent = "False" Abort.req { AbortIdentifier := "FAL", ReasonCode := "Invokeld-error-responding" }, Terminate.ind { LocallyGenerated := "True", Terminateldentifier := "FAL", ReasonCode := "Invokeld-error-responding" }, ResetArep	CLOSED

4.4.4 Functions

This subclause defines internal functions that are used by the AP AR Context state machine.

Table 19 – Function ResetArep

Name	ResetArep		
Input	Arep_Id	Output	True or False
Function	Closes the AP AR Context and initializes all elements of the AP attribute <i>List Of In-Use AR Endpoint Info</i> to zero (0).		

Table 20 – Function ApContextTest

Name	ApContextTest		
Input	Arep_Id	Output	True or False
Function	Locates the entry for the Selected AREP in the AP attribute <i>List Of In-Use AR Endpoint Info</i> , verifies its contents, and ensures that there is a matching AREP defined for the AR_ASE. The way in which the verification is carried out is dependent on implementation.		

Table 21 – Function ServicesSupportedTest

Name	ServicesSupportedTest		
Input	Arep_Id	Output	True or False
Function	Upon receipt of an Initiate-RequestPDU the called AP_ASE shall compare the Local List of Supported Services against those contained in the Initiate PDU. The ServicesSupportedTest fails if either the Local FAL does not support all the services as a responder that the remote FAL supports as a requestor, or the remote FAL does not support all the services as a responder that the local FAL supports as a requestor.		

Table 22 – Function ApExplicitConnection

Name	ApExplicitConnection		
Input	Arep_Id	Output	True or False
Function	Refer to AREP to determine if this AP_ASE supports explicit connection between APs.		

Table 23 – Function ImmediateAcknowledge

Name	ImmediateAcknowledge		
Input	Arep_Id	Output	True or False
Function	Refer to AREP to determine if this AP_ASE requires immediate acknowledge in the underlying layer.		

Table 24 – Function ConfirmedServiceCheck

Name	ConfirmedServiceCheck		
Input	Arep_Id	Output	True or False
Function	Determines if the called FAL service is a confirmed service except for AR ASE services.		

Table 25 – Function UnconfirmedServiceCheck

Name	UnconfirmedServiceCheck		
Input	Arep_Id	Output	True or False
Function	Determines if the called FAL service is an unconfirmed service except AR ASE services.		

Table 26 – Function ArServiceCheck

Name	ArServiceCheck		
Input	Arep_Id	Output	True or False
Function	Determines if the called service is an AR ASE service (FAL service or AR FSPM service).		

Table 27 – Function ArFspmService

Name	ArFspmService																						
Input	Arep_Id	Output	FAL Service or AR FSPM Service																				
Function	Generate an AR ASE service primitive. Relationship between FAL Services and AR FSPM Services shall be as follows: <table><tr><td>AR-Unconfirmed Send</td><td>UCS</td></tr><tr><td>AR-Confirmed Send</td><td>CS</td></tr><tr><td>AR-Establish</td><td>EST</td></tr><tr><td>AR-Abort</td><td>Abort</td></tr><tr><td>AR-Compel</td><td>FOMP</td></tr><tr><td>AR-Get Buffered Message</td><td>GBM</td></tr><tr><td>AR-Status</td><td>FSTS</td></tr><tr><td>AR-XON-OFF</td><td>AR_XON_OFF</td></tr><tr><td>AR-RemoteRead</td><td>RR</td></tr><tr><td>AR-RemoteWrite</td><td>RW</td></tr></table>			AR-Unconfirmed Send	UCS	AR-Confirmed Send	CS	AR-Establish	EST	AR-Abort	Abort	AR-Compel	FOMP	AR-Get Buffered Message	GBM	AR-Status	FSTS	AR-XON-OFF	AR_XON_OFF	AR-RemoteRead	RR	AR-RemoteWrite	RW
AR-Unconfirmed Send	UCS																						
AR-Confirmed Send	CS																						
AR-Establish	EST																						
AR-Abort	Abort																						
AR-Compel	FOMP																						
AR-Get Buffered Message	GBM																						
AR-Status	FSTS																						
AR-XON-OFF	AR_XON_OFF																						
AR-RemoteRead	RR																						
AR-RemoteWrite	RW																						

Table 28 – Function ArAcceeSupported

Name	ArAcceeSupported		
Input	Arep_Id	Output	True or False
Function	Determines if the AP accepts AR ASE Send (ConfirmedSend or UnconfirmedSend) services.		

Table 29 – Function MaxFalPduLengthTest

Name	MaxFalPduLengthTest		
Input	Arep_Id	Output	True or False
Function	Upon receipt of an Initiate-RequestPDU, the called AP_ASE shall check the requested maximum PDU length against its defined context. The following tests are performed: 1. If (ConfiguredMaxPDUSizeSending > MaxPDUSizeReceiving of the Initiate PDU), this test fails; 2. If (ConfiguredMaxPDUSizeReceiving < MaxPDUSizeSending of the Initiate PDU), this test fails.		

Table 30 – Function NegotiateOutstandingServices

Name	NegotiateOutstandingServices		
Input	Arep_Id	Output	True or False
Function	Upon receipt of an Initiate-RequestPDU, the called AP_ASE shall perform the following negotiation: - If ConfiguredMaxOutstandingServicesRequesting $> \text{MaxOutstandingServicesResponding}$ of the Initiate-RequestPDU then NegotiatedMaxOutstandingServicesRequesting $= \text{MaxOutstandingServicesResponding}$ of the Initiate-RequestPDU else NegotiatedMaxOutstandingServicesRequesting = ConfiguredMaxOutstandingServicesRequesting; - If ConfiguredMaxOutstandingServicesResponding $< \text{MaxOutstandingServicesRequesting}$ of the Initiate PDU then NegotiatedMaxOutstandingServicesResponding = ConfiguredMaxOutstandingServicesResponding else NegotiatedMaxOutstandingServicesResponding $= \text{MaxOutstandingServicesRequesting}$ of the Initiate PDU;		

Table 31 – Function RequestedServicesSupportedTest

Name	RequestedServicesSupportedTest		
Input	Arep_Id, Service.primitive	Output	True or False
Function	Upon receipt of a service request from FAL user, the called AP_ASE shall compare the Local List of Supported Services against the requested service primitive. This test fails if the service primitive is not contained in the Local List of Supported Services.		

Table 32 – Function IndicatedServicesSupportedTest

Name	IndicatedServicesSupportedTest		
Input	Arep_Id, service.primitive	Output	True or False
Function	Upon receipt of a service indication from AR_ASE, the called AP_ASE shall compare the Local List of Supported Services against the indicated service primitive. This test fails if the service primitive is not contained in the Local List of Supported Services.		

Table 33 – Function InvokeIdExistent

Name	InvokeIdExistent		
Input	Arep_Id, Invoke_Id	Output	True or False
Function	Upon receipt of a service primitive, the called AP_ASE shall - compare the local list of invoke IDs in use against the requested Invoke ID if the service primitive is request; - compare the local list of invoke IDs in use against the responded invoke ID if the service primitive is response. This test fails if the invoke ID does not exist in the local list of Invoke IDs in use.		

Table 34 – Function SameService

Name	SameService		
Input	Arep_Id, Invoke_Id	Output	True or False
Function	Upon receipt of a service primitive the called AP_ASE shall - compare the local list of outstanding services (responding) against the service if the service primitive is response; - compare the local list of outstanding services (requesting) against the service if the service primitive is confirmation. This test fails if the service is not the same as that in the local list of outstanding services.		

4.5 FAL Service Protocol Machine (FSPM)

The FAL Service Protocol Machine is common to all the AREP types. Only applicable primitives are different among different AREP types. It has one state called "ACTIVE."

4.5.1 Primitive Definitions

4.5.1.1 Primitives Exchanged between AP_Context and FSPM

Table 35 – Primitives issued by AP_Context to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST.req	AP_Context	Arep_Id, Data, Remote_DLCEP_Address	This primitive is used to convey an Establish request primitive from the AP_Context to the FSPM.
EST.rsp(+)	AP_Context	Arep_Id, Data	This primitive is used to convey an Establish response(+) primitive from the AP_Context to the FSPM.
EST.rsp(-)	AP_Context	Arep_Id, Data	This primitive is used to convey an Establish response(-) primitive from the AP_Context to the FSPM.
Abort.req	AP_Context	Arep_Id, Identifier, Reason_Code, Additional_Detail	This primitive is used to convey an Abort request primitive from the AP_Context to the FSPM.
CS.req	AP_Context	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) request primitive from the AP_Context to the FSPM.
CS.rsp	AP_Context	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) response primitive from the AP_Context to the FSPM.
UCS.req	AP_Context	Arep_Id, Remote_DLSAP_Address, Data	This primitive is used to convey an Unconfirmed Send (UCS) request primitive from the AP_Context to the FSPM.
FCMP.req	AP_Context	Arep_Id	This primitive is used to convey an FAL-Compel (FCMP) request primitive from the AP_Context to the FSPM.
GBM.req	AP_Context	Arep_Id	This primitive is used to convey a Get-Buffered-Message (GBM) request primitive from the AP_Context to the FSPM.
RW.req	AP_Context	Arep_Id, Data, Priority	This primitive is used to remotely write the contents of a buffer.
RR.req	AP_Context	Arep_Id, Priority	This primitive is used to get the contents of a remote buffer.
UCA.req	AP_Context	Arep_Id, Remote_Dlsap_Address, Data	This primitive is used to send an unconfirmed service request.
AR_XON_OFF.req	AP_Context	Arep_Id, Remote_Dlsap_Address, Data	This primitive is used to perform flow control.

Table 36 – Primitives issued by FSPM to AP_Context

Primitive Name	Source	Associated Parameters	Functions
EST.ind	FSPM	Arep_Id, Data	This primitive is used to convey an Establish indication primitive from the FSPM to the AP_Context.
EST.cnf(+)	FSPM	Arep_Id, Data	This primitive is used to convey an Establish result(+) primitive from the FSPM to the AP_Context.
EST.cnf(-)	FSPM	Arep_Id, Data	This primitive is used to convey an Establish result(-) primitive from the FSPM to the AP_Context.
Abort.ind	FSPM	Arep_Id, Locally_Generated, Identifier, Reason_Code, Additional_Detail	This primitive is used to convey an Abort indication primitive from the FSPM to the AP_Context.
CS.ind	FSPM	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) indication primitive from the FSPM to the AP_Context.
CS.cnf	FSPM	Arep_Id, Data	This primitive is used to convey a Confirmed Send (CS) confirmation primitive from the FSPM to the AP_Context.
UCS.ind	FSPM	Arep_Id, Remote_DLSAP_Address, Duplicate_FAL-SDU, Data, Local_Timeliness, Remote_Timeliness	This primitive is used to convey an Unconfirmed Send (UCS) indication primitive from the FSPM to the AP_Context.
FCMP.cnf	FSPM	Arep_Id, Status	This primitive is used to convey an FAL-Compel (FCMP) confirmation primitive from the FSPM to the AP_Context.
GBM.cnf(+)	FSPM	Arep_Id, Duplicate_FAL-SDU, Data, Local_Timeliness, Remote_Timeliness	This primitive is used to convey a Get-Buffered-Message (GBM) positive confirmation primitive from the FSPM to the AP_Context.
GBM.cnf(-)	FSPM	Arep_Id	This primitive is used to convey a Get-Buffered-Message (GBM) negative confirmation primitive from the FSPM to the AP_Context.
FSTS.ind	FSPM	Arep_Id, Reported_Status	This primitive is used to convey an FAL-Status (FSTS) indication primitive from the FSPM to the AP_Context.
UCA.ind	FSPM	Arep_Id, Remote_DLSAP_Address, Duplicate_FAL-SDU, Data	This primitive is used to convey an unconfirmed service indication.
AR_XON_OFF.ind	FSPM	Arep_Id, Remote_DLSAP_Address, Data	This primitive is used to convey a flow control indication.
RW.cnf(+)	FSPM	Arep_Id, Status	This primitive is used to convey a Remote_Write (RW) positive confirmation primitive.
RW.cnf(-)	FSPM	Arep_Id, Status	This primitive is used to convey a Remote_Write (RW) negative confirmation primitive.
RR.cnf(+)	FSPM	Arep_Id, Data, Local_Timeliness, Remote_Timeliness	This primitive is used to convey a Remote_Read (RR) positive confirmation primitive.
RR.cnf(-)	FSPM	Arep_Id, Status	This primitive is used to convey a Remote_Read (RR) negative confirmation primitive.

4.5.1.2 Parameters of AP_Context /FSPM Primitives

All the parameters used in the primitives exchanged between the AP_Context and the FSPM are identical to those defined in the “Operational Services” clause.

4.5.2 FSPM State Tables

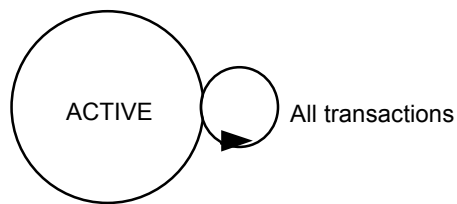


Figure 25 – State Transition Diagram of FSPM

Table 37 – FSPM State Table – Sender Transactions

#	Current State	Event Action	Next State
S1	ACTIVE	EST.req && SelectArep (Arep_Id) = "True" EST_req { user_data := Data, remote_dlcep_address := Remote_DLCEP_Address }	ACTIVE
S2	ACTIVE	EST.rsp(+) && SelectArep (Arep_Id) = "True" EST_rsp(+) { user_data := Data }	ACTIVE
S3	ACTIVE	EST.rsp(-) && SelectArep (Arep_Id) = "True" EST_rsp(-) { user_data := Data }	ACTIVE
S4	ACTIVE	UCS.req && SelectArep (Arep_Id) = "True" UCS_req { remote_dlsap_address := Remote_DLSAP_Address, user_data := Data }	ACTIVE
S5	ACTIVE	CS.req && SelectArep (Arep_Id) = "True" CS_req { user_data := Data }	ACTIVE
S6	ACTIVE	CS.rsp && SelectArep (Arep_Id) = "True" CS_rsp { user_data := Data }	ACTIVE
S7	ACTIVE	Abort.req && SelectArep (Arep_Id) = "True" Abort_req { identifier := Identifier, reason_code := Reason_Code, additional_detail := Additional_Detail }	ACTIVE
S8	ACTIVE	FCMP.req && SelectArep (Arep_Id) = "True" FCMP_req {	ACTIVE

#	Current State	Event Action	Next State
S9	ACTIVE	GBM.req && SelectArep (Arep_Id) = "True" GBM_req { }	ACTIVE
S10	ACTIVE	RW.req && SelectArep(Arep_Id) = "True" RW_req{ user_data := Data, priority := Priority }	ACTIVE
S11	ACTIVE	RR.req && SelectArep(Arep_Id) = "True" RR_req { priority := Priority }	ACTIVE
S12	ACTIVE	UCA.req && SelectArep (Arep_Id) = "True" UCA_req { remote_dlsap_address := Remote_DLSAP_Address, user_data := Data }	ACTIVE
S13	ACTIVE	AR_XON_OFF.req && SelectArep (Arep_Id) = "True" AR_XON_OFF_req { remote_dlsap_address := Remote_DLSAP_Address, user_data := Data }	ACTIVE
S14	ACTIVE	Any FSPM.req && SelectArep (Arep_Id) = "False" (no actions defined by the protocol, see notes 1 and 2.)	ACTIVE
NOTE1 A primitive generated in the FSPM sender state machine is sent to an appropriate ARPM that is selected by the FSPM using the SelectArep function. The Arep_Id parameter supplied by the AP_Context is the argument of this function. NOTE2 If the SelectArep function returns the value of False, it is a local matter to report such instance and the FSPM does not generate any primitives for the ARPM.			

Table 38 – FSPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	ACTIVE	EST_ind EST_ind { Arep_Id := arep_id, Data := user_data }	ACTIVE
R2	ACTIVE	EST_cnf(+) EST.cnf(+) { Arep_Id := arep_id, Data := user_data }	ACTIVE
R3	ACTIVE	EST_cnf(-) EST.cnf(-) { Arep_Id := arep_id, Data := user_data }	ACTIVE

#	Current State	Event Action	Next State
R4	ACTIVE	UCS_ind <pre> UCS.ind { Arep_Id := arep_id, Remote_DLSAP_Address := remote_dlsap_address, Duplicate_FAL-SDU := duplicate_fal_sdu, Data := user_data, Local_Timeliness := local_timeliness, Remote_Timeliness := remote_timeliness } </pre>	ACTIVE
R5	ACTIVE	CS_ind <pre> CS.ind { Arep_Id := arep_id, Data := user_data } </pre>	ACTIVE
R6	ACTIVE	CS_cnf <pre> CS.cnf { Arep_Id := arep_id, Data := user_data } </pre>	ACTIVE
R7	ACTIVE	Abort_ind <pre> Abort.ind { Arep_Id := arep_id, Locally_Generated := locally_generated, Identifier := identifier, Reason_Code := reason_code, Additional_Detail := additional_detail } </pre>	ACTIVE
R8	ACTIVE	FCMP_cnf <pre> FCMP.cnf { Arep_Id := arep_id, Status := status } </pre>	ACTIVE
R9	ACTIVE	GBM_cnf(+) <pre> GBM.cnf(+) { Arep_Id := arep_id, Duplicate_FAL-SDU := duplicate_fal_sdu, Data := user_data, Local_Timeliness := local_timeliness, Remote_Timeliness := remote_timeliness } </pre>	ACTIVE
R10	ACTIVE	GBM_cnf(-) <pre> GBM.cnf(-) { Arep_Id := arep_id, } </pre>	ACTIVE
R11	ACTIVE	FSTS_ind <pre> FSTS.ind { Arep_Id := arep_id, Reported_Status := reported_status } </pre>	ACTIVE
R12	ACTIVE	RW_cnf(-) <pre> RW.cnf(-) { Arep_Id := arep_id, Status := status } </pre>	ACTIVE
R13	ACTIVE	RW_cnf(+) <pre> RW.cnf(+) { Arep_Id := arep_id, Status := status } </pre>	ACTIVE

#	Current State	Event Action	Next State
R14	ACTIVE	RR_cnf(-) RR.cnf(-) { Arep_Id := arep_id, Status := status }	ACTIVE
R15	ACTIVE	RR_cnf(+) RR.cnf(+) { Arep_Id := arep_id, Data := user_data, Local_Timeliness := local_timeliness, Remote_Timeliness := remote_timeliness }	ACTIVE
R16	ACTIVE	UCA_ind UCA.ind { Arep_Id := arep_id, Remote_DLSAP_Address := remote_dlsap_address, Duplicate_FAL-SDU := duplicate_fal_sdu, Data := user_data, }	ACTIVE
R17	ACTIVE	AR_XON_OFF_ind AR_XON_OFF.ind { Arep_Id := arep_id, Remote_DLSAP_Address := remote_dlsap_address, Data := user_data, }	ACTIVE

4.5.2.1 Functions

Table 39 – Function SelectArep

Name	SelectArep	Used in	FSPM
Input		Output	
Arep_Id		True False	
Function	Looks for the AREP entry that is specified by the Arep_Id parameter. The Arep_Id parameter is provided with the AP_Context service primitives.		

4.6 Application Relationship Protocol Machines (ARPMs)

4.6.1 Queued Usertriggered Unidirectional (QUU) ARPM

4.6.1.1 Primitive Definitions

4.6.1.1.1 Primitives Exchanged between ARPM and FSPM

Table 40 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
UCS_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 41 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.
FSTS_ind	ARPM	arep_id, reported_status	This is an FAL internal primitive used to convey confirmation status.

4.6.1.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 42.

Table 42 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
Arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
User_data	This parameter conveys FAL-User data.
Locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
Identifier	This parameter conveys value that is used for the Identifier parameter.
Reason_code	This parameter conveys value that is used for the Reason_Code parameter.
Additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
Remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.

4.6.1.2.1 DLL Mapping of QUU AREP Class

This clause describes the mapping of the QUU AREP Class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data link layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUU AREP class are defined in this clause.

CLASS: QuuCl

PARENT CLASS: QueuedUser-TriggeredUnidirectionalAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDisapAddress
2	(c)	Constraint:	RemoteAddressConfigurationType = Linked
2.1	(m)	Attribute:	RemoteDisapAddress
3	(m)	Attribute:	LocalDisapRole (Basic, Group)
4	(c)	Constraint:	Role = ReportSource
4.1	(m)	Attribute:	DefaultQosAsSender
4.1.1	(m)	Attribute:	DIIPriority (Urgent, Normal, TimeAvailable)
4.1.2	(m)	Attribute:	MaxConfirmDelayOnUnitdata
4.1.3	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
4.1.4	(m)	Attribute:	DISchedulingPolicy (Implicit)
4.1.5	(o)	Attribute:	DleRemoteConf (True, False)
5	(m)	Attribute:	ExplicitQueue (True, False)
6	(c)	Constraint:	ExplicitQueue = True
6.1	(m)	Attribute:	QueueBindings—either for a sender or a receiver
6.1.1	(m)	Attribute:	MaxQueueDepth
6.1.2	(m)	Attribute:	MaxDisduSize

DLL SERVICES:

1	(m)	OpsService:	DL-Unitdata
2	(c)	Constraint:	ExplicitQueue = True
3	(m)	OpsService:	DL-Get

4.6.1.2.1.1 Attributes

LocalDisapAddress

This attribute specifies the DLSAP address to which this AREP is attached. This attribute is a DLSAP-address if the Role attribute has the value of ReportSource, and either a group DL-address or a DLSAP-address if the Role attribute has the value of ReportSink.

This attribute supplies the value for the “DL(SAP)-address” parameter specified in the DLL.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

RemoteDisapAddress

This attribute specifies the remote address to which FAL-PDUs are sent (for ReportSource AREPs), or from which they are received (for ReportSink AREPs).

If the RemoteAddressConfigurationType attribute is Linked, the value of this attribute has been configured. If it is Free, the value of this attribute is provided with a service request.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

LocalDisapRole

This attribute specifies the behavior of the local DLSAP to be used. If the Role is ReportSource, this attribute has the value of Basic. If the Role is ReportSink, it has the value of either Basic or Group.

This attribute supplies the value for the “DL(SAP)-role” parameter specified by the DLL.

DefaultQosAsSender

The DefaultQosAsSender attributes specify the DLL quality of service that is used by the sending AREP. The receiving DLL shall support the quality of service specified by these attributes.

DIIPriority

This attribute defines the DLL priority, and thus restricts the maximum length of an FAL-PDU, of the conveyance path of an AR.

This attribute supplies the value for the “DLL priority” parameter of the DLL. The values Urgent, Normal, and Time-Available correspond to URGENT, NORMAL, and TIME-AVAILABLE as defined in the Fieldbus Data Link Layer standard, respectively.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same QUU AREP.

MaxConfirmDelayOnUnitdata

This attribute specifies the maximum confirmation delay for a local confirmation from a DL-Unitdata request primitive.

This attribute supplies the value for the “Max confirm delay on locally-confirmed DL-Unitdata” parameter of the DLL.

DlpduAuthentication

This attribute specifies the lower bound of the length of the DL-addresses to be used by the DLL.

This attribute supplies the value for the “DLPDU authentication” parameter of the DLL. The values Ordinary, Source, and Maximal correspond to ORDINARY, SOURCE, and MAXIMAL as defined in the Fieldbus Data link layer standard, respectively.

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as it is passed from the FAL.

This attribute supplies the value for DL-Scheduling-policy attribute. Only the value Implicit is used. It corresponds to IMPLICIT as defined in the Fieldbus Data link layer standard.

DleRemoteConf

This optional attribute specifies, when it is present and true, that the remote immediate DLL acknowledgement shall be used. If it is not present, the remote immediate DLL acknowledgement shall not be used.

ExplicitQueue

This attribute specifies, when True, that the characteristics of the associated sending and receiving queues are explicitly configured and managed through the Network Management. The value of False means that queues with implementation specific depth and length are provided by the DLL.

QueueBindings

The following attributes specify the explicit queue that is bound to this DLSAP. For a sender, the queue is for sending. For a receiver, it is for receiving.

MaxQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued for sending or receiving.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL.

MaxDlsduSize

This attribute specifies the maximum length of an FAL-PDU that can be sent or received by the DLL.

This attribute supplies the value for the “Maximum DLSDU size” parameter of the DLL.

4.6.1.2.1.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.1.3 QUU ARPM State Machine

4.6.1.3.1 QUU ARPM States

The defined states and their descriptions of the QUU ARPM are listed below:

Table 43 – QUU ARPM States

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

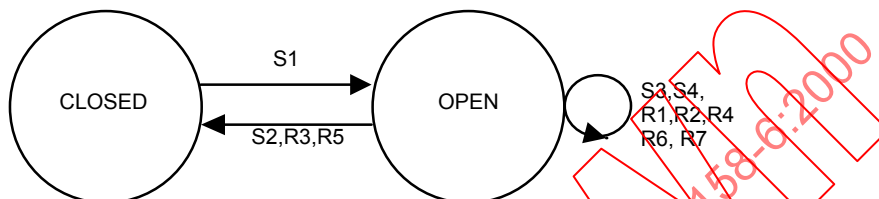


Figure 26 – State Transition Diagram of the QUU ARPM

4.6.1.3.2 QUU ARPM state table

Table 44 – QUU ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req EST_cnf(+) { arep_id := GetArepld (), user_data := "null" }	OPEN
S2	OPEN	Abort_req (no actions taken)	CLOSED
S3	OPEN	UCS_req && RemoteAddressConfigurationType = "Linked" && Role = "ReportSource" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_PDU", fal_data := user_data) } }	OPEN
S4	OPEN	UCS_req && RemoteAddressConfigurationType = "Free" && Role = "ReportSource" RemoteDlsapAddress := remote_dlsap_address, FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_PDU", fal_data := user_data) } }	OPEN

Table 45 – QUU ARPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && RemoteAddressConfigurationType = "Linked" && RemoteDisapAddress = calling_address && Role = "ReportSink" && FAL_Pdu_Type(fal_pdu) = "UCS_PDU" UCS_ind { arep_id := GetArepld (), user_data := fal_pdu }	OPEN
R2	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && RemoteAddressConfigurationType = "Free" && Role = "ReportSink" && FAL_Pdu_Type(fal_pdu) = "UCS_PDU" UCS_ind { arep_id := GetArepld (), remote_disap_address := calling_address, user_data := fal_pdu }	OPEN
R3	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && Role = "ReportSink" && FAL_Pdu_Type(fal_pdu) <> "UCS_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R4	OPEN	ErrorToARPM (no actions taken)	OPEN
R5	OPEN	FAL-PDU_ind && Role = "ReportSource" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid Event for Role", additional_detail := "null" }	CLOSED
R6	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_unidata_cnf" && Role = "ReportSource" && Fal_Pdu_Type(fal_pdu) = "UCS_PDU" && reason = "success" FSTS_Ind{ arep_id := GetArep(), reported_status := "dl_unidata_cnf" }	OPEN
R7	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_unidata_cnf" && Role = "ReportSource" && Fal_Pdu_Type(fal_pdu) = "UCS_PDU" && reason <> "success" FSTS_Ind{ arep_id := GetArep(), reported_status := "unidata_cnf NOACK" }	OPEN

4.6.1.3.3 Functions used by QUU ARPM

Table 46 – Function GetArepld ()

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 47 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_pdu_name, fal_data		disdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 48 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
fal_pdu		One of the FAL-PDU types defined in the FAL-PDUs section.	
Function	This function decodes the FAL-PDU that is conveyed in the fal_pdu parameter and retrieves one of the FAL-PDU types.		

4.6.2 Queued Usertriggered Bidirectional-Connection Oriented (QUB-CO) ARPM

4.6.2.1 Primitive Definitions

4.6.2.1.1 Primitives Exchanged between ARPM and FSPM

Table 49 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated parameters	Functions
EST_req	FSPM	user_data, remote_dlcep_address	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.

Table 50 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation primitive from the ARPM to the FSPM.

4.6.2.1.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 51.

Table 51 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
reported_status	This parameter conveys a Data Link Layer event status.
Remote_dlcep_address	This parameter conveys value of the remote dlcep address.

4.6.2.2 DLL Mapping of QUB AREP Class

This clause describes the mapping of the QUB AREP Class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data link layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB AREP class are defined in this clause.

CLASS: QubCo

PARENT CLASS: QueuedUser-TriggeredBidirectionalAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDlcepAddress
2	(c)	Constraint:	RemoteAddressConfigurationType = Linked
2.1	(m)	Attribute:	RemoteDlcepAddress
3	(c)	Constraint:	Role = Client Peer
3.1	(m)	Attribute:	Initiator (True, False)
4	(m)	Attribute:	DisapRole (Basic)
5	(m)	Attribute:	QosParameterSet
5.1	(m)	Attribute:	DlcepClass (Peer)
5.2	(m)	Attribute:	DlcepDataDeliveryFeatures
5.2.1	(m)	Attribute:	FromRequestorToResponder (Classical, Disordered)
5.2.2	(m)	Attribute:	FromResponderToRequestor (Classical, Disordered)
5.3	(m)	Attribute:	Priority
5.3.1	(m)	Attribute:	DllPriority (Urgent, Normal, TimeAvailable)
5.3.2	(m)	Attribute:	DllPriorityNegotiated (Urgent, Normal, TimeAvailable)
5.4	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
5.5	(m)	Attribute:	ResidualActivity
5.5.1	(m)	Attribute:	ResidualActivityAsSender (True, False)
5.5.2	(m)	Attribute:	ResidualActivityAsReceiver (True, False)
5.6	(m)	Attribute:	MaxConfirmDelay
5.6.1	(m)	Attribute:	MaxConfirmDelayOnDlConnect
5.6.2	(m)	Attribute:	MaxConfirmDelayOnDlData
5.7	(m)	Attribute:	DlSchedulingPolicy (Implicit)
5.8	(m)	Attribute:	ExplicitQueue (True, False)
5.9	(c)	Constraint:	ExplicitQueue = True
5.9.1	(m)	Attribute:	MaxDlsduSizes
5.9.1.1	(m)	Attribute:	MaxDlsduSizeFromRequestor
5.9.1.2	(m)	Attribute:	MaxDlsduSizeFromResponder
5.9.1.3	(m)	Attribute:	MaxDlsduSizeFromRequestorNegotiated
5.9.1.4	(m)	Attribute:	MaxDlsduSizeFromResponderNegotiated
5.9.2	(m)	Attribute:	MaxQueueDepth
5.9.2.1	(m)	Attribute:	MaxSendingQueueDepth
5.9.2.2	(m)	Attribute:	MaxReceivingQueueDepth

DLL SERVICES:

1	(m)	OpsService:	DL-Data
2	(c)	Constraint:	ExplicitQueue = True
2.1	(m)	OpsService:	DL-Get
3	(m)	OpsService:	DL-Connect
4	(m)	OpsService:	DL-Connection-Established
5	(m)	OpsService:	DL-Disconnect

4.6.2.2.1 Attributes

4.6.2.2.1.1 LocalDlcepAddress

This attribute specifies the local DLCEP address and identifies the DLCEP.

The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

NOTE1 The value of this attribute is also carried in the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

4.6.2.2.1.2 RemoteAddressConfigurationType

This attribute specifies how this AREP is used with the remote AREP. The value of “Free” means that this AREP can communicate with any remote AREP. The value of “Linked” means that this AREP can only communicate with the AREP specified by the RemoteDlcepAddress attribute.

RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP.

This attribute supplies the value for called DLCEP-address of the DL-Connect service.

NOTE The value of this attribute is also carried in the header part of the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

4.6.2.2.1.3 Role

This attribute specifies possible roles of this AREP. The value of "Client" means that this AREP issues FAL request PDUs to a remote AREP whose Role is either "Server" or "Peer." An AREP whose Role is "Server" is a mirror of "Client." An AREP whose Role is "Peer" can act as a "Client" and a "Peer" simultaneously.

Initiator

This attribute specifies whether this AREP is capable of issuing a DL-Connect request or not. The value of "True" means it is capable of doing it. If the value of this attribute is "False," it can only accept a DL-Connect indication from a remote AREP.

4.6.2.2.1.4 DisapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP.

This attribute supplies the value for the "DL(SAP)-role" parameter. The possible value Basic corresponds to BASIC as defined in the Data link layer standard.

4.6.2.2.1.5 QosParameterSet

The QosParameterSet attributes specify the DL quality of service that is used by this AREP. These attribute values may be negotiated with the remote AREP.

DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP.

This attribute supplies the value for the "DLCEP class" parameter of the DLL. The possible value of this attribute is Peer and corresponds to Peer defined by the DLL.

DlcepDataDeliveryFeatures

These two attributes specify data delivery features of the DLL.

The permitted values Classical and Disordered correspond respectively to CLASSICAL and DISORDERED defined by the Data link layer standard.

The FromRequestorToResponder and FromResponderToRequestor attributes shall have the same value.

FromRequestorToResponder

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of "True" to the remote AREP. It supplies the value for the "DLCEP data delivery features from requestor to responder(s)" parameter defined in the DLL.

FromResponderToRequestor

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of "False" to the remote AREP. It supplies the value for the "DLCEP data delivery features from responder(s) to requestor" parameter defined in the DLL.

Priority

DllPriority

This attribute specifies the configured value of the DLL priority.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same QUB AREP. Also, it is not permitted to use different priorities for the send and receive conveyance paths.

DllPriorityNegotiated

This attribute specifies the negotiated value of the DLL priority.

DlpduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the “DLPDU-authentication” parameter of the DLL. The permitted value Ordinary, Source, and Maximal correspond to ORDINARY, Source, and MAXIMAL, respectively, as defined in the Fieldbus Data link layer standard.

ResidualActivityAsSender

This attribute specifies sender’s DLC residual activity. It supplies the value for the “Residual activity as sender” parameter defined in the DLL. The possible values are “True” and “False.”

ResidualActivityAsReceiver

This attribute specifies receiver’s DLC residual activity. It supplies the value for the “Residual activity as receiver” parameter defined in the DLL. The possible values are “True” and “False.”

MaxConfirmDelay

This attribute specifies the maximum confirmation delay of certain DLL connection-oriented services.

MaxConfirmDelayOnDLConnect

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query” parameter specified in the Data link layer standard.

MaxConfirmDelayOnDLData

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Data service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Data” parameter specified in the Data link layer standard (IEC 61158-3).

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as possible.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Implicit corresponds to IMPLICIT defined in the Fieldbus Data link layer standard.

ExplicitQueue

MaxDlsduSizeFromRequestor

This attribute specifies the configured value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

MaxDlsduSizeFromResponder

This attribute specifies the configured value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from responder” parameter of the DLL.

MaxDlsduSizeFromRequestorNegotiated

This attribute specifies the negotiated value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

MaxDlsduSizeFromResponderNegotiated

This attribute specifies the negotiated value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

MaxSendingQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued for transmission.

This attribute supplies the value for the "Maximum queue depth" parameter of the DLL for Send queues.

MaxReceivingQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued at reception.

This attribute supplies the value for the "Maximum queue depth" parameter of the DLL for Receive queues.

4.6.2.2.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.2.3 QUB AREP State Machine

4.6.2.3.1 QUB ARPM States

The defined states and their descriptions of the QUB ARPM are listed below:

Table 52 – QUB ARPM States

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs. It may send or receive Establish service FAL-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
REQUESTING (REQ)	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING (RSP)	The AREP has received an Establish Request FAL-PDU, delivered an Establish.ind primitive and is waiting for a response from its user.
REPLIED (REPL)	The Server AREP has issued an EST_rsp(+) primitive and is waiting for receiving a "connection-established" indication from the DLL.
SAME	Indicates that the next state is the same as the current state.

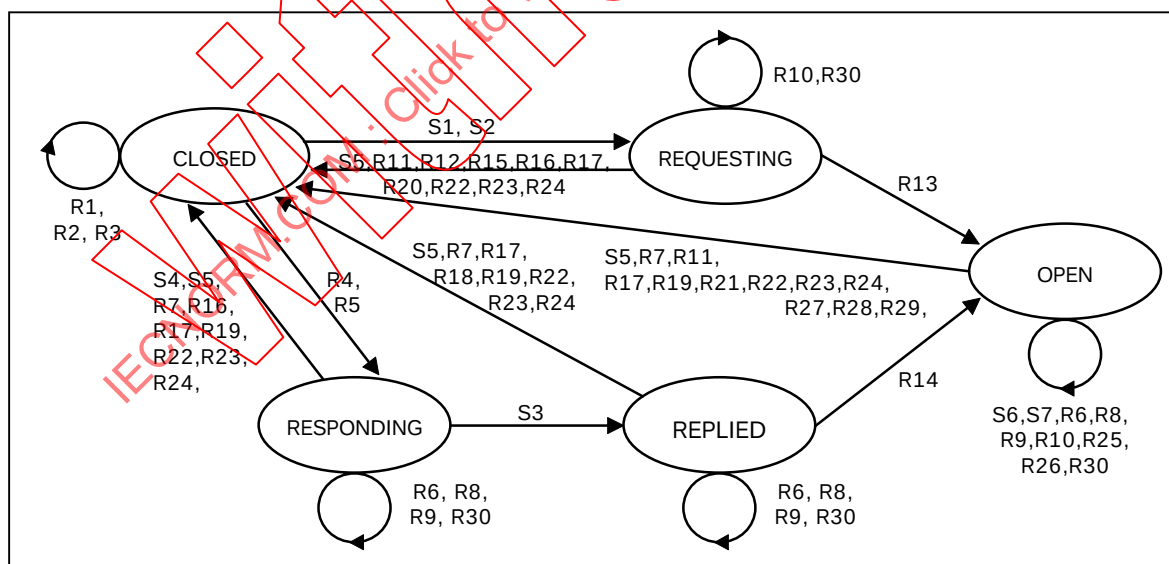


Figure 27 – State Transition Diagram of QUB ARPM

4.6.2.3.2 QUB ARPM state table

Table 53 – QUB ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType := "Free" RemoteDlcepAddress := remote_dlcep_address, FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) }	REQ
S2	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType := "Linked" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) }	REQ
S3	RSP	EST_rsp(+) FAL-PDU_req { dmpm_service_name := "DMPM_Connect_rsp", arep_id := GetArepld (), responding_address := "default dlsap address", local_dlcep_address := LocalDlcepAddress, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_RspPDU", fal_data := user_data) }	REPL
S4	RSP	EST_rsp(-) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "connection rejection–transient condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ErrPDU", fal_data := user_data) }	CLOSED
S5	NOT CLOSED	Abort_req FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection–normal condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) }	CLOSED

#	Current State	Event Action	Next State
S6	OPEN	CS_req && Role = "Client" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_ReqPDU", fal_data := user_data) } }	OPEN
S7	OPEN	CS_rsp && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_RspPDU", fal_data := user_data) } }	OPEN

Table 54 – QUB ARPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	CLOSED	Connect_ind && Initiator = "True" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Multiple Initiators", dlsdu := "null" }	CLOSED
R2	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) <> "EST_ReqPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }	CLOSED
R3	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Remote Address Mismatch", dlsdu := "null" }	CLOSED
R4	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" RemoteDlcepAddress := calling_dlcep_address, MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id := GetArepld (), user_data := dls_user_data }	RSP

#	Current State	Event Action	Next State
R5	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress = calling_dlcep_address MaxDisduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDisduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id := GetArepld (), user_data := dls_user_data }	RSP
R6	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Remote Address Mismatch", dlsdu := "null" }	SAME
R7	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU" }	CLOSED
R8	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "AREP Busy", dlsdu := "null" }	SAME
R9	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) <> "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }	SAME

#	Current State	Event Action	Next State
R10	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Multiple Initiators", dlsdu := "null" }	SAME
R11	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Multiple Initiators", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Multiple Initiators" }	CLOSED
R12	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) <> "EST_RspPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU" }	CLOSED
R13	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) = "EST_RspPDU" MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, DllPriorityNegotiated := dll_priority, EST_cnf(+) { arep_id := GetArepld (), user_data := dls_user_data }	OPEN
R14	REPL	FAL-PDU_ind && dmpm_service_name = "DMPM_Connection_Established_ind" (no actions taken)	OPEN
R15	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && FAL_Pdu_Type (fal_pdu) = "EST_ErrPDU" EST_cnf(-) { arep_id := GetArepld (), user_data := fal_pdu }	CLOSED

#	Current State	Event Action	Next State
R16	REQ RSP	FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event" }	CLOSED
R17	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (fal_pdu) = "Abort_PDU" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := AbortIdentifier (fal_pdu), reason_code := AbortReason (fal_pdu), additional_detail := AbortDetail (fal_pdu) }	CLOSED
R18	REPL	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DM_Connection_Established_ind")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True" identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" }	CLOSED
R19	REPL RSP OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (fal_pdu) <> "Abort_PDU" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R20	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && ((FAL_Pdu_Type (fal_pdu) <> "Abort_PDU") && (FAL_Pdu_Type (fal_pdu) <> "EST_ErrPDU")) Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED

#	Current State	Event Action	Next State
R21	OPEN	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DMPM_Data_ind")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" } 	CLOSED
R22	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" } 	CLOSED
R23	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_user" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := "FAL", reason_code := reason, additional_detail := "null" } 	CLOSED
R24	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" } 	CLOSED
R25	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Peer" "Server" && FAL_Pdu_Type (fal_pdu) = "CS_ReqPDU" CS_ind { arep_id := GetArepld (), user_data := fal_pdu } 	OPEN
R26	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" "Peer" && FAL_Pdu_Type (fal_pdu) = "CS_RspPDU" CS_cnf { arep_id := GetArepld (), user_data := fal_pdu } 	OPEN

#	Current State	Event Action	Next State
R27	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Server" && FAL_Pdu_Type (fal_pdu) <> "CS_ReqPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R28	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" && FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R29	OPEN	FAL-PDU_ind && Role = "Peer" && dmpm_service_name = "DMPM_Data_ind" && ((FAL_Pdu_Type (fal_pdu) <> "CS_ReqPDU") && (FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R30	NOT CLOSED	ErrorToARPM (No actions taken. See NOTE below.) NOTE It is a local matter to report this error status to network management entities. The ARPM does not abort the existing connections. The FAL user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive.	SAME

4.6.2.3.3 Functions used by QUB ARPM**Table 55 – Function GetArepld ()**

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 56 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail		disdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 57 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data fal_pdu		One of the FAL-PDU types defined in clause 4.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data or fal_pdu parameter and retrieves one of the FAL-PDU types.		

Table 58 – Function AbortIdentifier

Name	AbortIdentifier	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 59 – Function AbortReason

Name	AbortReason	Used in	ARPM
Input		Output	
fal_pdu		The Reason Code parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter.		

Table 60 – Function AbortDetail

Name	AbortDetail	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.		

4.6.3 Queued Usertriggered Bidirectional-Connectionless (QUB-Cl) ARPM

4.6.3.1 Primitive Definitions

4.6.3.1.1 Primitives Exchanged between ARPM and FSPM

Table 61 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
UCS_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 62 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, remote_dlsap_address, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.
FSTS_ind	ARPM	arep_id, reported_status	This is an FAL internal primitive used to convey a FAL-Status (FSTS) indication primitive from the ARPM to the FSPM.

4.6.3.1.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 63.

Table 63 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL-SDU parameter.
remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Link Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.3.2 DLL Mapping of QUB-CL AREP Class

This clause describes the mapping of the QUB_CL AREP Class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data link layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB_CL AREP class are defined in this clause.

CLASS: QubCl

PARENT CLASS: QueuedUser-TriggeredBidirectionalConnectionlessAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDisapAddress
2	(m)	Attribute:	RemoteDisapAddress
3	(m)	Attribute:	LocalDisapRole (Basic)
4	(m)	Attribute:	DefaultQosAsSender
4.1	(m)	Attribute:	DllPriority (Urgent, Normal, TimeAvailable)
4.2	(m)	Attribute:	MaxConfirmDelayOnUnitdata
4.3	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
4.4	(m)	Attribute:	DlSchedulingPolicy (Implicit)
4.5	(m)	Attribute:	DleRemoteConf (True, False)
5	(m)	Attribute:	ExplicitQueue (True, False)
6	(c)	Constraint:	ExplicitQueue = True
6.1	(m)	Attribute:	SendingQueueBindings
6.1.1	(m)	Attribute:	MaxQueueDepth
6.1.2	(m)	Attribute:	MaxDisduSize
6.2	(m)	Attribute:	ReceivingQueueBindings
6.2.1	(m)	Attribute:	MaxQueueDepth
6.2.2	(m)	Attribute:	MaxDisduSize

DLL SERVICES:

1	(m)	OpsService:	DL-Unitdata
2	(c)	Constraint:	ExplicitQueue = True
2.1	(m)	OpsService:	DL-Get

4.6.3.2.1 Attributes

4.6.3.2.1.1 LocalDisapAddress

This attribute specifies the DLSAP address to which this AREP is attached. It supplies the value for the "DL(SAP)-address" parameter specified in the DLL.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

4.6.3.2.1.2 RemoteDisapAddress

This attribute specifies the remote address to which FAL-PDUs are sent (for Source AREPs), or from which they are received (for Sink AREPs).

If the RemoteAddressConfigurationType attribute is Linked, the value of this attribute has been configured. If it is Free, the value of this attribute is provided with a service request.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

4.6.3.2.1.3 LocalDisapRole

This attribute specifies the behavior of the local DLSAP to be used. No DLCEPs are needed on this Basic DLSAPRole.

This attribute supplies the value for the "DL(SAP)-role" parameter specified by the DLL.

4.6.3.2.1.4 DefaultQosAsSender

The DefaultQosAsSender attributes specify the DLL quality of service that is used by the sending AREP. The receiving DLL shall support the quality of service specified by these attributes.

DllPriority

This attribute defines the DLL priority, and thus restricts the maximum length of an FAL-PDU, of the conveyance path of an AR.

This attribute supplies the value for the "DLL priority" parameter of the DLL. The values Urgent, Normal, and Time-Available correspond to URGENT, NORMAL, and TIME-AVAILABLE as defined in the Fieldbus Data link layer standard, respectively.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same QUB_CL AREP. The priority is identical on the sending way and on the receiving way.

MaxConfirmDelayOnUnitdata

This attribute specifies the maximum confirmation delay for a local confirmation from a DL-Unitdata request primitive.

This attribute supplies the value for the “Max confirm delay on locally-confirmed DL-Unitdata” parameter of the DLL.

DlpduAuthentication

This attribute specifies the lower bound of the length of the DL-addresses to be used by the DLL.

This attribute supplies the value for the “DLPDU authentication” parameter of the DLL. The values Ordinary, Source, and Maximal correspond to ORDINARY, SOURCE, and MAXIMAL as defined in the Fieldbus Data link layer standard, respectively.

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as it is passed from the FAL.

This attribute supplies the value for DL-Scheduling-policy attribute. Only the value Implicit is used. It corresponds to IMPLICIT as defined in the Fieldbus Data link layer standard.

DleRemoteConf

This attribute specifies, when true, that the remote DLL immediate Ack shall be used.

4.6.3.2.1.5 ExplicitQueue

This attribute specifies, when True, that the characteristics of the associated sending and receiving queues are explicitly configured and managed through the Network Management. The value of False means that queues with implementation specific depth and length are provided by the DLL.

QueueBindings

The following attributes specify the explicit queue that is bound to this DLSAP. For the sending part, the queue is for sending. For the receiving part, it is for receiving.

MaxQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued for sending or receiving.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL.

MaxDisduSize

This attribute specifies the maximum length of an FAL-PDU that can be sent or received by the DLL.

This attribute supplies the value for the “Maximum DLSDU size” parameter of the DLL.

4.6.3.2.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.3.3 QUB-CL ARPM State Machine

4.6.3.3.1 QUB-CL ARPM States

The defined states and their descriptions of the QUB-CL ARPM are listed below:

Table 64 – QUB-CL ARPM States

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

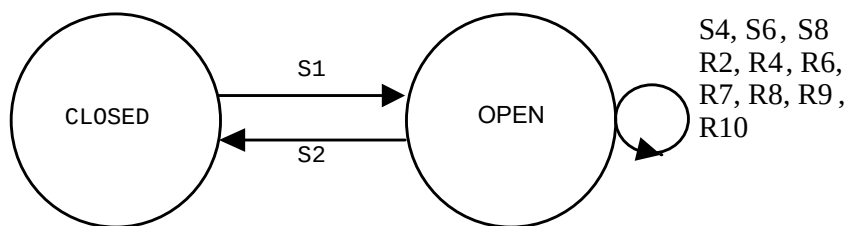


Figure 28 – State Transition Diagram of the QUB-CL ARPM

4.6.3.3.2 QUB-CL ARPM State Table

Table 65 – QUB-CL ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req EST_cnf(+) { arep_id := GetArepld (), user_data := "null" }	OPEN
S2	OPEN	Abort_req (no action taken)	CLOSED
S3	OPEN	CS_req && ConfigurationType = "Linked" && Role = "Client" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), called_address := Remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_ReqPDU", fal_data := user_data) } }	OPEN
S4	OPEN	CS_req && RemoteAddressConfigurationType = "Free" && Role = "Client" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), called_address := remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_ReqPDU", fal_data := user_data) } }	OPEN
S5	OPEN	CS_rsp && ConfigurationType = "Linked" && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), called_address := Remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_RspPDU", fal_data := user_data) } }	OPEN

#	Current State	Event Action	Next State
S6	OPEN	CS_rsp && RemoteAddressConfigurationType = "Free" && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), called_address := remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_RspPDU", fal_data := user_data) } 	OPEN
S7	OPEN	UCS_req && ConfigurationType = "Linked" && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), called_address := Remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_PDU", fal_data := user_data) } 	OPEN
S8	OPEN	UCS_req && RemoteAddressConfigurationType = "Free" && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Unitdata_req", arep_id := GetArepld (), called_address := remote_dlsap_address, dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_PDU", fal_data := user_data) } 	OPEN

Table 66 – QUB-CI ARPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && ConfigurationType = "Linked" && RemoteDlsapAddress = calling_address && FAL_Pdu_Type (fal_pdu) = "CS_ReqPDU" && Role = "Server" "Peer" CS_ind { arep_id := GetArepld (), remote_dlsap_address := "null", user_data := fal_pdu } 	OPEN
R2	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && RemoteAddressConfigurationType = "Free" && FAL_Pdu_Type (fal_pdu) = "CS_ReqPDU" && Role = "Server" "Peer" CS_ind { arep_id := GetArepld (), remote_dlsap_address := calling_address, user_data := fal_pdu } 	OPEN

#	Current State	Event Action	Next State
R3	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && ConfigurationType = "Linked" && RemoteDlsapAddress = calling_address && FAL_Pdu_Type (fal_pdu) = "CS_RspPDU" && Role = "Client" "Peer" CS_cnf { arep_id := GetArepld (), remote_dlsap_address := "null", user_data := fal_pdu }	OPEN
R4	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && RemoteAddressConfigurationType = "Free" && FAL_Pdu_Type (fal_pdu) = "CS_RspPDU" && Role = "Client" "Peer" CS_cnf { arep_id := GetArepld (), remote_dlsap_address := calling_address, user_data := fal_pdu }	OPEN
R5	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && ConfigurationType = "Linked" && RemoteDlsapAddress = calling_address && FAL_Pdu_Type (fal_pdu) = "UCS_PDU" && Role = "Server" "Peer" UCS_ind { arep_id := GetArepld () remote_dlsap_address := "null", user_data := fal_pdu }	OPEN
R6	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_ind" && RemoteAddressConfigurationType = "Free" && FAL_Pdu_Type (fal_pdu) = "UCS_PDU" && Role = "Client" "Peer" UCS_ind { arep_id := GetArepld (), remote_dlsap_address := calling_address, user_data := fal_pdu }	OPEN
R7	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Unitdata_cnf" && FALPdu_Type(fal_pdu) = "CS_Req PDU" && Role = "Client Peer" && dl-status <> success CS_Cnf { arep_id := GetArepld(), remote_dlsap_address := null if Linked otherwise calling_address, user_data := null, result := dl_status }	OPEN
R8	OPEN	FAL-PDU_Ind && dmpm_service_name = "DMPM_Unitdata_cnf" && FALPdu_Type(fal_pdu) = "CS_Req PDU" && Role = "Client Peer" && dl-status = "success" (no action)	OPEN

#	Current State	Event Action	Next State
R9	OPEN	FAL-PDU_Ind && dmpm_service_name = "DMPM_Unidata_cnf" && FALPdu_Type(fal-pdu) = "UCS_Req PDU" && Role = "Client Peer" FSTS_Ind{ arep_Id := GetArepId() reported_status := dl_status }	OPEN
R10	OPEN	FAL-PDU_Ind && dmpm_service_name = "DMPM_Unidata_cnf" && FALPdu_Type(fal-pdu) = "CS_Rsp PDU" && Role = "Server Peer" (no action)	OPEN

4.6.3.3.3 Functions used by QUB-CI ARPM

Table 67 – Function GetArepId ()

Name	GetArepId ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 68 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail		dlsdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 69 – Function FAL-Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data		One of the FAL-PDU types defined in clause 4.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FAL-PDU types.		

4.6.4 Queued Usertriggered Bidirectional-Segmentation (QUB-Seg) ARPM

4.6.4.1 Primitive Definitions

4.6.4.1.1 Primitives Exchanged between ARPM and FSPM

Table 70 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
UCS_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 71 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, remote_dlsap_address, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.

4.6.4.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 72.

Table 72 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL-SDU parameter.
remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Link Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.4.2.1 DLL Mapping of QUB-Seg AREP Class

This clause describes the mapping of the QUB-Seg AREP class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data link layer standard; rather, it defines how they are used by this AREP class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB-Seg AREP class are defined in this clause.

This class is defined to support the concurrent use of confirmed and unconfirmed services on the same AR as well as the conveyance of services with a maximum data size of 64 kbytes by providing a segmentation functionality, which is invisible to the FAL user. This class uses the DDL features of CLASSICAL connections to ensure the correct segmentation and reassembling of FAL services.

CLASS: QubSeg

PARENT CLASS: QueuedUser-TriggeredBidirectionalSegmentationAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
3	(m)	Attribute:	DlsapRole (Basic)
4	(m)	Attribute:	QosParameterSet
4.1	(m)	Attribute:	DlcepClass (Peer)
4.2	(m)	Attribute:	DlcepDataDeliveryFeatures
4.2.1	(m)	Attribute:	FromRequestorToResponder (Classical)
4.2.2	(m)	Attribute:	FromResponderToRequestor (Classical)
4.3	(m)	Attribute:	Priority
4.3.1	(m)	Attribute:	DllPriority (Urgent, Normal, TimeAvailable)
4.3.2	(m)	Attribute:	DllPriorityNegotiated (Urgent, Normal, TimeAvailable)
4.4	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
4.5	(m)	Attribute:	ResidualActivity
4.5.1	(m)	Attribute:	ResidualActivityAsSender (True, False)
4.5.2	(m)	Attribute:	ResidualActivityAsReceiver (True, False)
4.6	(m)	Attribute:	MaxConfirmDelay
4.6.1	(m)	Attribute:	MaxConfirmDelayOnDllConnect
4.6.2	(m)	Attribute:	MaxConfirmDelayOnDllData
4.7	(m)	Attribute:	DlschedulingPolicy (Implicit)
4.8	(m)	Attribute:	ExplicitQueue (True, False)
4.9	(c)	Constraint:	ExplicitQueue = True
4.9.1	(m)	Attribute:	MaxDlsduSizes
4.9.1.1	(m)	Attribute:	MaxDlsduSizeFromRequestor
4.9.1.2	(m)	Attribute:	MaxDlsduSizeFromResponder
4.9.1.3	(m)	Attribute:	MaxDlsduSizeFromRequestorNegotiated
4.9.1.4	(m)	Attribute:	MaxDlsduSizeFromResponderNegotiated
4.9.2	(m)	Attribute:	QueueBindings
4.9.2.1	(m)	Attribute:	SendingBufferOrQueueIdentifier
4.9.2.2	(m)	Attribute:	ReceivingBufferOrQueueIdentifier
4.9.3	(m)	Attribute:	MaxQueueDepth
4.9.3.1	(m)	Attribute:	MaxSendingQueueDepth
4.9.3.2	(m)	Attribute:	MaximumReceivingQueueDepth

DLL SERVICES:

1	(m)	OpsService:	DL-Data
2	(c)	Constraint:	ExplicitQueue = True
2.1	(m)	OpsService:	DL-Get
3	(m)	OpsService:	DL-Connect
4	(m)	OpsService:	DL-Connection-Established
5	(m)	OpsService:	DL-Disconnect

4.6.4.2.1.1 Attributes

LocalDlcepAddress

This attribute specifies the local DLCEP address and identifies the DLCEP.

The value of this attribute is used as the "DLCEP-address" parameter of the DLL.

NOTE1 The value of this attribute is also carried in the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP.

This attribute supplies the value for called DLCEP-address of the DL-Connect service.

NOTE The value of this attribute is also carried in the header part of the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

DisapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP.

This attribute supplies the value for the “DL(SAP)-role” parameter. The possible value Basic corresponds to BASIC as defined in the Data link layer standard.

QosParameterSet

The QosParameterSet attributes specify the DL quality of service that is used by this AREP. These attribute values may be negotiated with the remote AREP.

DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP.

This attribute supplies the value for the “DLCEP class” parameter of the DLL. The possible value of this attribute is Peer and corresponds to Peer defined by the DLL.

DlcepDataDeliveryFeatures

These two attributes specify data delivery features of the DLL.

The permitted values Classical and Disordered correspond, respectively to CLASSICAL and DISORDERED defined by the Data link layer standard.

The FromRequestorToResponder and FromResponderToRequestor attributes shall have the same value.

FromRequestorToResponder

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP. It supplies the value for the “DLCEP data delivery features from requestor to responder(s)” parameter defined in the DLL. Only the value “CLASSICAL” is allowed since disordered segments would corrupt the FAL PDUs.

FromResponderToRequestor

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP. It supplies the value for the “DLCEP data delivery features from responder(s) to requestor” parameter defined in the DLL. Only the value “CLASSICAL” is allowed since disordered segments would corrupt the FAL PDUs.

Priority

DllPriority

This attribute specifies the configured value of the DLL priority.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same QUB AREP. Also, it is not permitted to use different priorities for the send and receive conveyance paths.

DllPriorityNegotiated

This attribute specifies the negotiated value of the DLL priority.

Dl pduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the “DLPDU-authentication” parameter of the DLL. The permitted value Ordinary, Source, and Maximal correspond to ORDINARY, Source, and MAXIMAL, respectively, as defined in the Fieldbus Data link layer standard.

ResidualActivityAsSender

This attribute specifies sender’s DLC residual activity. It supplies the value for the “Residual activity as sender” parameter defined in the DLL. The possible values are “True” and “False.”

ResidualActivityAsReceiver

This attribute specifies receiver’s DLC residual activity. It supplies the value for the “Residual activity as receiver” parameter defined in the DLL. The possible values are “True” and “False.”

MaxConfirmDelay

This attribute specifies the maximum confirmation delay of certain DLL connection-oriented services.

MaxConfirmDelayOnDLConnect

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query” parameter specified in the Data link layer standard.

MaxConfirmDelayOnDLData

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Data service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Data” parameter specified in the Data link layer standard (IEC 61158-3).

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as possible.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Implicit corresponds to IMPLICIT defined in the Fieldbus Data link layer standard.

ExplicitQueue

MaxDisduSizeFromRequestor

This attribute specifies the configured value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

MaxDisduSizeFromResponder

This attribute specifies the configured value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from responder” parameter of the DLL.

MaxDisduSizeFromRequestorNegotiated

This attribute specifies the negotiated value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

MaxDisduSizeFromResponderNegotiated

This attribute specifies the negotiated value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

SendingBufferOrQueueIdentifier

This attribute provides a local means to identify a queue that is used to store sending FAL-PDUs.

This attribute supplies the value for the “Buffer-and-queue bindings as sender” parameter of the DLL.

ReceivingBufferOrQueueIdentifier

This attribute provides a local means to identify a queue that is used to store receiving FAL-PDUs.

This attribute supplies the value for the “Buffer-and-queue bindings as receiver” parameter of the DLL.

MaxSendingQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued for transmission.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL for Send queues.

MaxReceivingQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued at reception.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL for Receive queues.

4.6.4.2.1.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.4.3 QUB-Seg ARPM State Machine

4.6.4.3.1 QUB-Seg ARPM States

The defined states and their descriptions of the QUB-Seg ARPM are listed below:

Table 73 – QUB-Seg ARPM States

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs. It may send or receive Establish service FAL-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
REQUESTING (REQ)	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING (RSP)	The AREP has received an Establish Request FAL-PDU, delivered an Establish.indication primitive and is waiting for a response from its user.
REPLIED (REPL)	The Server AREP has issued an EST_rsp(+) primitive and is waiting for receiving a "connection-established" indication from the DLL.
SAME	Indicates that the next state is the same as the current state.

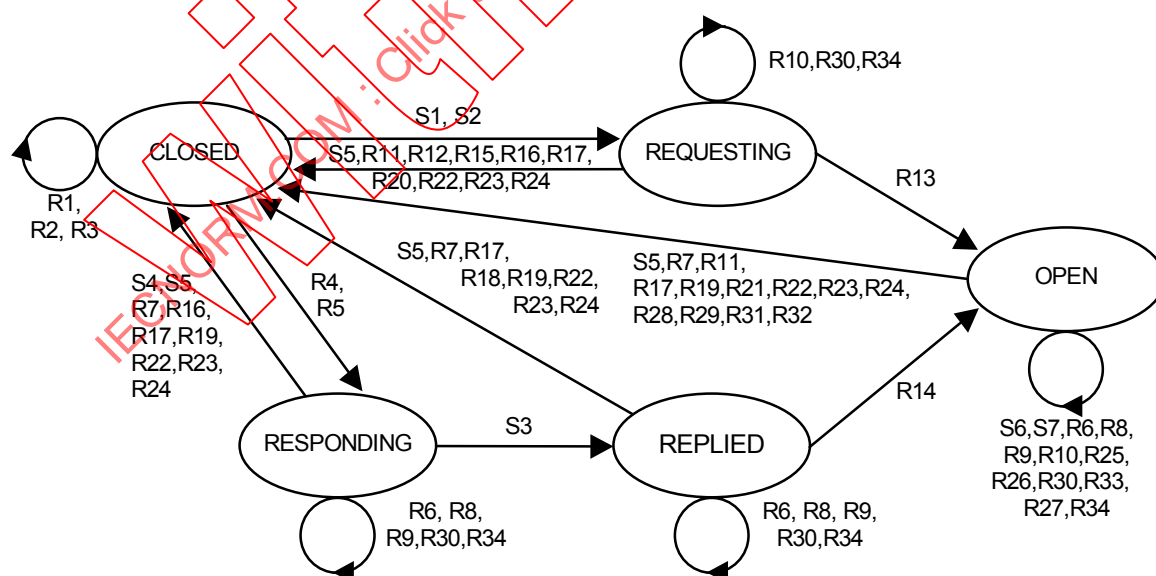


Figure 29 – State Transition Diagram of QUB-Seg ARPM

4.6.4.3.2 QUB-Seg ARPM State Table

Table 74 – QUB-Seg ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	<pre> EST_req && Initiator = "True" && RemoteAddressConfigurationType := "Free" RemoteDlcepAddress := remote_dlcep_address, FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) } </pre>	REQ
S2	CLOSED	<pre> EST_req && Initiator = "True" && RemoteAddressConfigurationType := "Linked" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) } </pre>	REQ
S3	RSP	<pre> EST_rsp(+) FAL-PDU_req { dmpm_service_name := "DMPM_Connect_rsp", arep_id := GetArepld (), responding_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_RspPDU", fal_data := user_data) } </pre>	REPL
S4	RSP	<pre> EST_rsp(-) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "connection rejection-transient condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ErrPDU", fal_data := user_data) } </pre>	CLOSED
S5	NOT CLOSED	<pre> Abort_req FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) } </pre>	CLOSED

#	Current State	Event Action	Next State
S6	OPEN	CS_req && Role = "Client" "Peer" && PDU length < MaxDisduSizeFromRequestorNegotiated FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_ReqPDU", fal_data := user_data) }	OPEN
S7	OPEN	CS_rsp && Role = "Server" "Peer" && PDU length < MaxDisduSizeFromRequestorNegotiated FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_RspPDU", fal_data := user_data) }	OPEN
S8	OPEN	UCS_req && Role = "Server" "Peer" && PDU length < MaxDisduSizeFromRequestorNegotiated FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_ReqPDU", fal_data := user_data) }	OPEN
S9	OPEN	CS_req && Role = "Client" "Peer" && PDU length >= MaxDisduSizeFromRequestorNegotiated for (n := 1 to (PDU length - 1) div (MaxDisduSizeFromRequestorNegotiated-1)+1) FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-Segment (fal_pdu_name := "CS_ReqPDU", fal_data := Segment_data, n) } endfor NOTE1 When the length of the FAL-PDU is greater or equal to max size Disdu negotiated, the QUB_Seg protocol splits the FAL-PDU into N ordered segment_data numbered from 1 to N. The first Segment_data contain the FAL_Header. NOTE2 For each Segment_data, the function "Build-FAL-Segment" builds an APDU-Segment := FAL Header followed with Segment_data without any gap. The header shall have its bit 4 := 1 for the Segment_data 1 to N-1, and 0 for the Segment_data N. NOTE3 The first APDU-Segment to be sent through the the network shall contain the Segment_data 1 and so on.	OPEN

#	Current State	Event Action	Next State
S10	OPEN	CS_rsp && Role = "Server" "Peer" && PDU length >= MaxDlsduSizeFromRequestorNegotiated for (n := 1 to (PDU length - 1) div (MaxDlsduSizeFromRequestorNegotiated-1)+1) FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-Segment (fal_pdu_name := "CS_RspPDU", fal_data := Segment_data, n) } endfor NOTE1 When the length of the FAL-PDU is greater or equal to max size Dlsdu negotiated, the QUB_Seg protocol splits the FAL-PDU into N ordered segment_data numbered from 1 to N. The first Segment_data shall contain the FAL_Header. NOTE2 For each segment_data, the function "Build-FAL-Segment" builds an APDU-Segment := FAL Header followed with Segment_data without any gap. The header shall have its bit 4 := 1 for the segment_data 1 to N-1, and 0 for the segment_data N. NOTE3 The first APDU-Segment to be sent through the network shall contain the Segment_data 1 and so on.	OPEN
S11	OPEN	UCS_req && Role = "Server" "Peer" && PDU length >= MaxDlsduSizeFromRequestorNegotiated for (n := 1 to (PDU length - 1) div (MaxDlsduSizeFromRequestorNegotiated-1)+1) FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-Segment (fal_pdu_name := "UCS_ReqPDU", fal_data := Segment_data, n) } endfor NOTE1 When the length of the FAL-PDU is greater or equal to max size Dlsdu negotiated, the QUB_Seg protocol splits the FAL-PDU into N ordered Segment_data numbered from 1 to N. The first Segment_data shall contain the FAL_Header. NOTE2 For each Segment_data, the function "Build-FAL-Segment" builds an APDU-Segment := FAL Header followed with Segment_data without any gap. The header shall have its bit 4 := 1 for the Segment_data 1 to N-1, and 0 for the Segment_data N. NOTE3 The first APDU-Segment to be sent through the the network contains the Segment_data 1 and so on.	OPEN

Table 75 – ARPM state table - Receiver transactions

#	Current State	Event Action	Next State
R1	CLOSED	Connect_ind && Initiator = "True" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Multiple Initiators", dlsdu := "null" }	CLOSED
R2	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) <> "EST_ReqPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }	CLOSED

#	Current State	Event Action	Next State
R3	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Remote Address Mismatch", dlsdu := "null" }	CLOSED
R4	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" RemoteDlcepAddress := calling_dlcep_address, MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id := GetArepld (), user_data := dls_user_data }	RSP
R5	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress = calling_dlcep_address MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id := GetArepld (), user_data := dls_user_data }	RSP
R6	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Remote Address Mismatch", dlsdu := "null" }	SAME
R7	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU" }	CLOSED

#	Current State	Event Action	Next State
R8	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "AREP Busy", dlsdu := "null" }	SAME
R9	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) <> "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }	SAME
R10	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Multiple Initiators", dlsdu := "null" }	SAME
R11	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Multiple Initiators", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Multiple Initiators" }	CLOSED
R12	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) <> "EST_RspPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU" }	CLOSED

#	Current State	Event Action	Next State
R13	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) = "EST_RspPDU" ResetIntermediatePDU () MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, DllPriorityNegotiated := dll_priority, EST_cnf(+) { arep_id := GetArepld (), user_data := dls_user_data }	OPEN
R14	REPL	FAL-PDU_ind && dmpm_service_name = "DMPM_Connection_Established_ind" ResetIntermediatePDU ()	OPEN
R15	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && FAL_Pdu_Type (fal_pdu) = "EST_ErrPDU" EST_cnf(-) { arep_id := GetArepld (), user_data := dls_user_data }	CLOSED
R16	REQ RSP	FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event" }	CLOSED
R17	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (fal_pdu) = "Abort_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "False", identifier := AbortIdentifier (fal_pdu), reason_code := AbortReason (fal_pdu), additional_detail := AbortDetail (fal_pdu) }	CLOSED

#	Current State	Event Action	Next State
R18	REPL	<pre> FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DM_Connection_Established_ind")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", disdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" } </pre>	CLOSED
R19	REPL RSP OPEN	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (fal_pdu) <> "Abort_PDU" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" } </pre>	CLOSED
R20	REQ	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && ((FAL_Pdu_Type (fal_pdu) <> "Abort_PDU") && (FAL_Pdu_Type (fal_pdu) <> "EST_ErrPDU")) Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" } </pre>	CLOSED
R21	OPEN	<pre> FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DMPM_Data_ind")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", disdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" } </pre>	CLOSED

#	Current State	Event Action	Next State
R22	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" }	CLOSED
R23	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_user" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := "FAL", reason_code := reason, additional_detail := "null" }	CLOSED
R24	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" }	CLOSED
R25	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Peer" "Server" && FAL_Pdu_Type (fal_pdu) = "CS_ReqPDU" && AddSegment (fal_pdu) = "OK" if (MoreFollows(fal_pdu) = "False") CS_ind { arep_id := GetArepld (), user_data := GetIntermediatePDU () } endif NOTE When the length of an FAL-PDU is greater or equal to the max size Dlsdu negotiated, the DLL delivers the received APDU segments, starting with the APDU segment 1 and ending with the APDU segment N (the same order as they were sent by the sender). Each APDU_Segment contains an FAL header and a Segment_data. The DLL disconnects the channel if a DLL frame (segment) is missing, the QUB_Seg protocol reassembles the Segment_data of the N received APDU segments. The N-1 APDU_Segments shall have a Header with bit 4 = 1, and the last APDU_Segment shall have a header with bit 4 = 0. The function "AddSegment" removes the header of the APDU_Segment and appends the Segment_data to the previous received Segment_data. Once the N Segment_data are appended together without any gap, the function "GetintermediatePDU" gives the original FAL-APDU.	OPEN

#	Current State	Event Action	Next State
R26	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" "Peer" && FAL_Pdu_Type (fal_pdu) = "CS_RspPDU" && AddSegment (fal_pdu) = "OK" if (MoreFollows(fal_pdu) = "False") CS_cnf { arep_id := GetArepld (), user_data := fal_pdu } endif NOTE When the length of an FAL-PDU is greater or equal to the max size Dlsdu negotiated, the DLL delivers the received APDU segments, starting with the APDU segment 1 and ending with the APDU segment N (the same order as they were sent by the sender). Each APDU_Segment contains a FAL header and a Segment_data. The DLL disconnects the channel if a DLL frame (segment) is missing, the QUB_Seg protocol reassembles the Segment_data of the N received APDU segments. The N-1 APDU_Segments shall have a Header with bit 4 = 1, and the last APDU_Segment shall have a header with bit 4 = 0. The function "AddSegment" removes the header of the APDU_Segment and appends the Segment_data to the previous received Segment_data. Once the N Segment_data are appended together without any gap, the function "GetintermediatePDU" gives the original FAL-APDU.	OPEN
R27	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Server" "Peer" && FAL_Pdu_Type (fal_pdu) = "UCS_ReqPDU" && AddSegment (fal_pdu) = "OK" if (MoreFollows(fal_pdu) = "False") UCS_ind { arep_id := GetArepld (), remote_dlsap_address = "null", user_data := fal_pdu } endif NOTE When the length of an FAL-PDU is greater or equal to the max size Dlsdu negotiated, the DLL delivers the received APDU segments, starting with the APDU segment 1 and ending with the APDU segment N (the same order as they were sent by the sender). Each APDU_Segment contains a FAL header and a Segment_data. The DLL disconnects the channel if a DLL frame (segment) is missing, the QUB_Seg protocol reassembles the Segment_data of the N received APDU segments. The N-1 APDU_Segments shall have a Header with bit 4 = 1, and the last APDU_Segment shall have a header with bit 4 = 0. The function "AddSegment" removes the header of the APDU_Segment and appends the Segment_data to the previous received Segment_data. Once the N Segment_data are appended together without any gap, the function "GetintermediatePDU" gives the original FAL-APDU.	OPEN
R28	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Peer" "Server" && FAL_Pdu_Type (fal_pdu) = "CS_ReqPDU UCS_ReqPDU" && AddSegment (fal_pdu) <> "OK" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED

#	Current State	Event Action	Next State
R29	OPEN	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Peer" "Client" && (FAL_Pdu_Type (fal_pdu) = "CS_RspPDU") && AddSegment (fal_pdu) <> "OK" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" } </pre>	CLOSED
R30	OPEN	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Data_Cnf(-)" && Role = "Server" && FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" } </pre>	CLOSED
R31	OPEN	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Data_Cnf" && Role = "Client" && FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" } </pre>	CLOSED

#	Current State	Event Action	Next State
R32	OPEN	FAL-PDU_ind && Role = "Peer" && dmpm_service_name = "DMPM_Data_ind" && ((FAL_Pdu_Type (fal_pdu) <> "CS_ReqPDU") && (FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU") && (FAL_Pdu_Type (fal_pdu) <> "UCS_ReqPDU")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" } }	CLOSED
R33	OPEN	FAL-PDU_Ind && dmpm-service-name = "DMPM_Data_Cnf(-)" && Role = "Client Peer" && FAL_Pdu_Type(fal_pdu) = "UCS_ReqPDU" FAL_PDU_Req{ dmpm-service-name := "DMPM_Disconnect-req" arep-Id := GetArepld(), reason := "dl-status", dl-sddu := "null" } } Abort_Ind{ arep_Id := GetArepld() locally-generated := "true" Identifier := "Data Link Layer" reason code := reason additional_detail := "null" } }	OPEN
R34	NOT CLOSED	ErrorToARPM (no actions taken) NOTE1: It is a local matter to report this error status to network management entities. The ARPM does not abort the existing connections. The FAL user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive. NOTE2: This event mainly occurs when the DLL has not been able to convey a DL service to the remote partner within the specified time interval. In this case an implementation may also decide to pass negative responses for all confirmed FAL service requests that are affected by this failure to the respective FAL user.	SAME

4.6.4.3.3 Functions used by QUB-Seg ARPM

Table 76 – Function GetArepld

Name	GetArepld	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 77 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
	fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail	dlsdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 78 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data		One of the FAL-PDU types defined in the FAL-PDUs section.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FAL-PDU types.		

Table 79 – Function AbortIdentifier

Name	AbortIdentifier	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 80 – Function AbortReason

Name	AbortReason	Used in	ARPM
Input		Output	
fal_pdu	The Reason Code parameter of the Abort service.		
Function			
This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter.			

Table 81 – Function AbortDetail

Name	AbortDetail	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.		

Table 82 – Function BuildFAL-Segment

Name	BuildFAL-Segment	Used in	ARPM
Input		Output	
	fal_pdu_name, fal_data, segment_number	dlsdu	
Function	Builds an FAL-Segment out of the parameters given as input variables. The segment_number must be ≥ 1 and defines, which part of the fal_pdu is to be put in the returned segment. This function adds the FAL-Header to each segment. It sets bit 4 of the FAL-Header to 1, indicating that there are more segments to be sent, when it is not the last segment. Otherwise bit 4 is set to 0. Except for this bit the FAL-Header is identical in all segments of one FAL-PDU.		

Table 83 – Function MoreFollows

Name	MoreFollows	Used in	ARPM
Input		Output	
dls_user_data		Boolean value	
Function	This function inspects the FAL-PDU that is conveyed in the dls_user_data parameter and returns "True", if bit5 of the FAL-Header is set. Otherwise it returns "False".		

NOTE The following three functions make use of a persistent variable IntermediatePDU, in which all segments of the current FAL User Data are stored by the receiver of this data.

Table 84 – Function ResetIntermediatePDU

Name	ResetIntermediatePDU	Used in	ARPM
Input		Output	
(none)		(none)	
Function	This function removes all segments which have so far been collected from the IntermediatePDU.		

Table 85 – Function AddSegment

Name	AddSegment	Used in	ARPM
Input		Output	
dls_user_data		Error Code	
Function	This function adds the received dls_user_data as a new segment to the IntermediatePDU. It checks segments reassembling protocol error. If no error was detected during these checks, the error code "OK" is returned.		

Table 86 – Function GetIntermediatePDU

Name	GetIntermediatePDU	Used in	ARPM
Input		Output	
(none)		fal_user_data	
Function	This function returns the complete FAL user data, which was received in multiple segments. After this function call the variable IntermediatePDU does not contain any segments.		

4.6.5 Queued Usertriggered Bidirectional-Flow Control (QUB-FC) ARPM

4.6.5.1 QUB-FC Primitive Definitions

4.6.5.1.1 Primitives Exchanged between ARPM and FSPM

Table 87 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
UCA_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Acknowledged Send (UCA) request primitive from the FSPM to the ARPM.
AR_XON_OFF_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey AR-XON-OFF request primitive from the FSPM to the ARPM.

Table 88 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation primitive from the ARPM to the FSPM.
UCA_ind	ARPM	arep_id, remote_dlsap_address, duplicate_fal_sdu, user_data	This is an FAL internal primitive used to convey an Unconfirmed Acknowledged Send (UCA) indication primitive from the ARPM to the FSPM.
AR_XON_OFF_ind	ARPM	arep_id, remote_dlsap_address, user_data	This is an FAL internal primitive used to convey a AR-XON-OFF indication primitive from the ARPM to the FSPM.

4.6.5.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 89.

Table 89 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
User_data	This parameter conveys FAL-User data.
Locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
Identifier	This parameter conveys value that is used for the Identifier parameter.
Reason_code	This parameter conveys value that is used for the Reason_Code parameter.
Additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
Duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL_SDU parameter.
Remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.
Status	This parameter conveys value that is used for the Status parameter.
Reported_status	This parameter conveys a Data Link Layer event status.
Local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
Remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.5.2.1 DLL Mapping of QUB-FC AREP Class

This clause describes the mapping of the QUB-FC AREP class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data link layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the QUB-FC AREP class are defined in this clause.

CLASS: QubFC

PARENT CLASS: QueuedUser-TriggeredBidirectional-FlowControlAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
3	(m)	Attribute:	DisapRole (Basic)
4	(m)	Attribute:	QosParameterSet
4.1	(m)	Attribute:	DlcepClass (Peer)
4.2	(m)	Attribute:	DlcepDataDeliveryFeatures
4.2.1	(m)	Attribute:	FromRequestorToResponder (Classical, Disordered)
4.2.2	(m)	Attribute:	FromResponderToRequestor (Classical, Disordered)
4.3	(m)	Attribute:	Priority
4.3.1	(m)	Attribute:	DlIPriority (Urgent, Normal, TimeAvailable)
4.3.2	(m)	Attribute:	DlIPriorityNegotiated (Urgent, Normal, TimeAvailable)
4.4	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
4.5	(m)	Attribute:	ResidualActivity
4.5.1	(m)	Attribute:	ResidualActivityAsSender (True, False)
4.5.2	(m)	Attribute:	ResidualActivityAsReceiver (True, False)
4.6	(m)	Attribute:	MaxConfirmDelay
4.6.1	(m)	Attribute:	MaxConfirmDelayOnDIConnect
4.6.2	(m)	Attribute:	MaxConfirmDelayOnDIData
4.7	(m)	Attribute:	DlSchedulingPolicy (Implicit)
4.8	(m)	Attribute:	ExplicitQueue (True, False)
4.9	(c)	Constraint:	ExplicitQueue = True
4.9.1	(m)	Attribute:	MaxDisduSizes
4.9.1.1	(m)	Attribute:	MaxDisduSizeFromRequestor
4.9.1.2	(m)	Attribute:	MaxDisduSizeFromResponder
4.9.1.3	(m)	Attribute:	MaxDisduSizeFromRequestorNegotiated
4.9.1.4	(m)	Attribute:	MaxDisduSizeFromResponderNegotiated
4.9.2	(m)	Attribute:	MaxQueueDepth
4.9.2.1	(m)	Attribute:	MaxSendingQueueDepth
4.9.2.2	(m)	Attribute:	MaximimReceivingQueueDepth

DLL SERVICES:

1	(m)	OpsService:	DL-Data
2	(c)	Constraint:	ExplicitQueue = True
2.1	(m)	OpsService:	DL-Get
3	(m)	OpsService:	DL-Connect
4	(m)	OpsService:	DL-Connection-Established
5	(m)	OpsService:	DL-Disconnect

4.6.5.2.1.1 Attributes

LocalDlcepAddress

This attribute specifies the local DLCEP address and identifies the DLCEP.

The value of this attribute is used as the "DLCEP-address" parameter of the DLL.

NOTE1 The value of this attribute is also carried in the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP. This attribute supplies the value for called DLCEP-address of the DL-Connect service.

NOTE The value of this attribute is also carried in the header part of the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

DisapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP. This attribute supplies the value for the "DL(SAP)-role" parameter. The possible value Basic corresponds to BASIC as defined in the Data link layer standard.

QosParameterSet

The QosParameterSet attributes specify the DL quality of service that is used by this AREP. These attribute values may be negotiated with the remote AREP.

DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP. This attribute supplies the value for the "DLCEP class" parameter of the DLL. The possible value of this attribute is Peer and corresponds to Peer defined by the DLL.

DlcepDataDeliveryFeatures

These two attributes specify data delivery features of the DLL. The permitted values Classical and Disordered correspond, respectively to CLASSICAL and DISORDERED defined by the Data link layer standard.

The FromRequestorToResponder and FromResponderToRequestor attributes shall have the same value.

FromRequestorToResponder

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of "True" to the remote AREP. It supplies the value for the "DLCEP data delivery features from requestor to responder(s)" parameter defined in the DLL.

FromResponderToRequestor

This attribute specifies the DLL data delivery feature of the DLPDUs sent from the AREP whose Initiator attribute has a value of "False" to the remote AREP. It supplies the value for the "DLCEP data delivery features from responder(s) to requestor" parameter defined in the DLL.

Priority

DllPriority

This attribute specifies the configured value of the DLL priority.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same QUB AREP. Also, it is not permitted to use different priorities for the send and receive conveyance paths.

DllPriorityNegotiated

This attribute specifies the negotiated value of the DLL priority.

DlpduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the "DLPDU-authentication" parameter of the DLL. The permitted value Ordinary, Source, and Maximal correspond to ORDINARY, Source, and MAXIMAL, respectively, as defined in the Fieldbus Data link layer standard.

ResidualActivity

ResidualActivityAsSender

This attribute specifies the sender's DLC residual activity. It supplies the value for the "Residual activity as sender" parameter defined in the DLL. The possible values are "True" and "False."

ResidualActivityAsReceiver

This attribute specifies the receiver's DLC residual activity. It supplies the value for the "Residual activity as receiver" parameter defined in the DLL. The possible values are "True" and "False."

MaxConfirmDelay

This attribute specifies the maximum confirmation delay of certain DLL connection-oriented services.

MaxConfirmDelayOnDLConnect

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the "Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query" parameter specified in the Data link layer standard.

MaxConfirmDelayOnDLData

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Data service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Data” parameter specified in the Data Link Layer standard.

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as possible.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Implicit corresponds to IMPLICIT defined in the Data Link Layer standard.

ExplicitQueue**MaxDlsduSize****MaxDlsduSizeFromRequestor**

This attribute specifies the configured value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

MaxDlsduSizeFromResponder

This attribute specifies the configured value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from responder” parameter of the DLL.

MaxDlsduSizeFromRequestorNegotiated

This attribute specifies the negotiated value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “True” to the remote AREP.

MaxDlsduSizeFromResponderNegotiated

This attribute specifies the negotiated value of the maximum length of an FAL-PDU that can be sent from the AREP whose Initiator attribute has a value of “False” to the remote AREP.

MaxQueueDepth**MaxSendingQueueDepth**

This attribute specifies the maximum number of FAL-PDUs that can be queued for transmission.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL for Send queues.

MaxReceivingQueueDepth

This attribute specifies the maximum number of FAL-PDUs that can be queued at reception.

This attribute supplies the value for the “Maximum queue depth” parameter of the DLL for Receive queues.

4.6.5.2.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.5.3 QUB-FC ARPM State Machine**4.6.5.3.1 QUB-FC ARPM States**

The defined states and their descriptions of the QUB-FC ARPM are listed below:

Table 90 – QUB-FC ARPM States

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs. It may send or receive Establish service FAL-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
REQUESTING (REQ)	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING (RSP)	The AREP has received an Establish Request FAL-PDU, delivered an Establish.indication primitive and is waiting for a response from its user.
REPLIED (REPL)	The Server AREP has issued an EST_rsp(+) primitive and is waiting for receiving a "connection-established" indication from the DLL.
SAME	Indicates that the next state is the same as the current state.

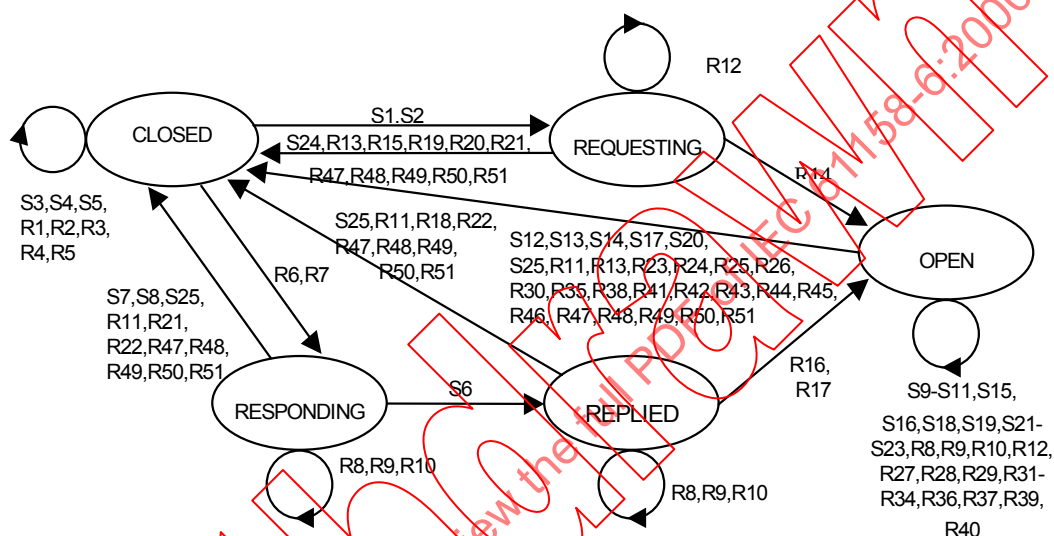


Figure 30 – State Transition Diagram of QUB-FC ARPM

4.6.5.3.2 QUB-FC ARPM State Table

Table 91 – QUB-FC ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType := "Free" RemoteDlcepAddress := remote_dlcep_address, FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) } 	REQ

#	Current State	Event Action	Next State
S2	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType := "Linked" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) }	REQ
S3	CLOSED	EST_req && Initiator = "False" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid Event for Role" }	CLOSED
S4	CLOSED	unallowed primitive Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed AREP primitive in connection establishment " }	CLOSED
S5	CLOSED	Abort_req ignore	CLOSED
S6	RSP	EST_rsp(+) FAL-PDU_req { dmpm_service_name := "DMPM_Connect_rsp", arep_id := GetArepld (), responding_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_RspPDU", fal_data := user_data) }	REPL
S7	RSP	EST_rsp(-) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "connection rejection-transient condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ErrPDU", fal_data := user_data) }	CLOSED
S8	RSP	unallowed primitive Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed AREP primitive in connection establishment " }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := " Unallowed AREP primitive in connection establishment ", dlsdu := "null" }	CLOSED

#	Current State	Event Action	Next State
S9	OPEN	CS_req && Role = "Client" "Peer" && Xon="True" && GetCounterValue(OSCC) < maxOSCC && CIU = 0 FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_ReqPDU", fal_data := user_data) }, IncrementCounter(OSCC)	OPEN
S10	OPEN	CS_req && Role = "Client" "Peer" && Xon="True" && GetCounterValue(OSCC) < maxOSCC && CIU > 0 FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_ReqPDU", fal_data := user_data) }, StartTimer(TC) IncrementCounter(OSCC)	OPEN
S11	OPEN	CS_req && Role = "Client" "Peer" && Xon="False" CS_cnf (-) { arep_id := GetArepld (), user_data := "null" }	OPEN
S12	OPEN	CS_req && Role = "Client" "Peer" && Xon="True" && GetCounterValue(OSCC) ≥ maxOSCC Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Number of parallel services exceeded" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Number of parallel services exceeded", dlsdu := "null" }, StopTimer ResetCounters	CLOSED

#	Current State	Event Action	Next State
S13	OPEN	CS_req && Role = "Server" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed service as server" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := " Unallowed service as server", dlsdu := "null" }, StopTimer ResetCounters	CLOSED
S14	OPEN	CS_rsp && Role = "Server" "Peer" && CIU = 0 FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_RspPDU", fal_data := user_data) }, DecrementCounter(OSCS)	OPEN
S15	OPEN	CS_rsp && Role = "Server" "Peer" && CIU > 0 FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "CS_RspPDU", fal_data := user_data) }, StartTimer(TS) DecrementCounter(OSCS)	OPEN
S16	OPEN	AR_XON_OFF_req && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "AR_XON_OFF_ReqPDU", fal_data := user_data) }	OPEN

#	Current State	Event Action	Next State
S17	OPEN	CS_rsp && Role = "Client" "Peer" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed service as client" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := " Unallowed service as client", dlsdu := "null" }, StopTimer ResetCounters	CLOSED
S18	OPEN	UCA_req && Xon = "True" && Role = "Client" "Peer" && GetCounterValue(UCC) < maxUCC && CIU = 0 FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCA_PDU", fal_data := user_data) }, IncrementCounter(UCC)	OPEN
S19	OPEN	UCA_req && Xon = "True" && Role = "Client" "Peer" && GetCounterValue(UCC) < maxUCC && CIU > 0 FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCA_PDU", fal_data := user_data) }, StartTimer(TC) IncrementCounter(UCC)	OPEN
S20	OPEN	UCA_req && Xon = "True" && Role = "Server" "Peer" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed service as server" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := " Unallowed service as server", dlsdu := "null" }, StopTimer ResetCounters	CLOSED

#	Current State	Event Action	Next State
S21	OPEN	UCA_req && Xon = "False" (no actions taken)	OPEN
S22	OPEN	TCTimer expired StartTimer(TC) FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "IdlePDU", fal_data := "null") } }	OPEN
S23	OPEN	TSTimer expired StartTimer(TS) FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "IdlePDU", fal_data := "null") } }	OPEN
S24	NOT CLOSED	Abort_req FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) }, StopTimer, ResetCounters	CLOSED

Table 92 – QUB-FC ARPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	CLOSED	Connect_ind && Initiator = "True" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Multiple Initiators", dlsdu := "null" }	CLOSED
R2	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) <> "EST_ReqPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Invalid FAL-PDU", dlsdu := "null" }	CLOSED

#	Current State	Event Action	Next State
R3	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && AREPContextCheck (dlsdu) = "False" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Context Check Negative", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := "Context Check Negative", fal_additional_detail := maxOSCC, maxOSCS, maxUCSC, maxUCSS, CI) } }	CLOSED
R4	CLOSED	unallowed primitive (no actions taken)	CLOSED
R5	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Remote Address Mismatch", dlsdu := "null" } }	CLOSED
R6	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && AREPContextCheck (dlsdu) = "True" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress = calling_dlcep_address MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id := GetArepld (), user_data := dls_user_data } }	RSP
R7	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && AREPContextCheck (dlsdu) = "True" && RemoteAddressConfigurationType = "Free" RemoteDlcepAddress := calling_dlcep_address, MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id := GetArepld (), user_data := dls_user_data } }	RSP
R8	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Remote Address Mismatch", dlsdu := "null" } }	SAME

#	Current State	Event Action	Next State
R9	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "AREP Busy", dlsdu := "null" }	SAME
R10	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) <> "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Invalid FAL-PDU", dlsdu := "null" }	SAME
R11	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_Type (dls_user_data) = "EST_ReqPDU" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU" }	CLOSED
R12	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Multiple Initiators", dlsdu := "null" }	SAME

#	Current State	Event Action	Next State
R13	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), dlcep_dl_id := dlcep_dl_id (from Connect_ind), reason := "Multiple Initiators", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Multiple Initiators" }	CLOSED
R14	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) = "EST_RspPDU" MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, DlIPriorityNegotiated := dll_priority, EST_cnf(+) { arep_id := GetArepld (), user_data := dls_user_data }	OPEN
R15	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) <> "EST_RspPDU" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU" }	CLOSED
R16	REPL	FAL-PDU_ind && dmpm_service_name = "DMPM_Connection_Established_ind" && CIU=0 (no actions taken)	OPEN
R17	REPL	FAL-PDU_ind && dmpm_service_name = "DMPM_Connection_Established_ind" && CIU>0 StartTimer(RS)	OPEN

#	Current State	Event Action	Next State
R18	REPL	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DM_Connection_Established_ind")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" }	CLOSED
R19	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && FAL_Pdu_Type (dls_user_data) = "EST_ErrPDU" EST_cnf(-) { arep_id := GetArepld (), user_data := dls_user_data }	CLOSED
R20	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && ((FAL_Pdu_Type (dls_user_data) <> "Abort_PDU") && (FAL_Pdu_Type (dls_user_data) <> "EST_ErrPDU")) Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R21	REQ RSP	FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event" }	CLOSED
R22	REPL RSP	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (dls_user_data) <> "Abort_PDU" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED

#	Current State	Event Action	Next State
R23	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (dls_user_data) <> "Abort_PDU" && Role = "Client" II "Peer" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R24	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (dls_user_data) <> "Abort_PDU" && Role = "Server" II "Peer" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R25	OPEN	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") II (dmpm_service_name <> "DMPM_Data_ind")) && Role = "Client" II "Peer" FAL-PDU_req{ dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED

#	Current State	Event Action	Next State
R26	OPEN	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") (dmpm_service_name <> "DMPM_Data_ind")) && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }, Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DL-Event", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R27	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Peer" "Server" && FAL_Pdu_Type (dls_user_data) = "CS_ReqPDU" && GetCounterValue(OSCS) < maxOSCS && CIU = 0 CS_ind { arep_id := GetArepld (), user_data := fal_pdu }, IncrementCounter(OSCS)	OPEN
R28	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Peer" "Server" && FAL_Pdu_Type (dls_user_data) = "CS_ReqPDU" && GetCounterValue(OSCS) < maxOSCS && CIU > 0 CS_ind { arep_id := GetArepld (), user_data := fal_pdu }, IncrementCounter(OSCS) , StartTimer(RS)	OPEN
R29	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" "Peer" && FAL_Pdu_Type (dls_user_data) = "AR_XON_OFF_ReqPDU" SetXon (user_data), AR_XON_OFF_ind { arep_id := GetArepld (), user_data := fal_pdu }	OPEN

#	Current State	Event Action	Next State
R30	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Server" "Peer" && FAL_Pdu_Type (dls_user_data) = "CS_ReqPDU" && GetCounterValue(OSCS) ≥ maxOSCS FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Number of parallel services exceeded", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Number of parallel services exceeded", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R31	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" "Peer" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && CIU = 0 CS_cnf { arep_id := GetArepld (), user_data := fal_pdu }, DecrementCounter(OSCC)	OPEN
R32	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" "Peer" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && CIU > 0 CS_cnf { arep_id := GetArepld (), user_data := fal_pdu }, DecrementCounter(OSCC), StartTimer(RC)	OPEN
R33	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "UCA_ReqPDU" && Role = "Server" "Peer" && CIU = 0 UCA_ind { arep_id := GetArepld (), user_data := fal_pdu }, FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCA_AckPDU", fal_data := "null") }	OPEN

#	Current State	Event Action	Next State
R34	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "UCA_ReqPDU" && Role = "Server" II "Peer" && CIU > 0 UCA_ind { arep_id := GetArepld (), user_data := fal_pdu }, FAL-PDU_req { dmpm_service_name := "DMPM_Data_req", arep_id := GetArepld (), disdu := BuildFAL-PDU (fal_pdu_name := "UCA_AckPDU", fal_data := "null") }, StartTimer(RS)	OPEN
R35	OPEN	FAL-PDU_ind && dmpm_service_name = DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "UCA_ReqPDU" && Role = "Client" II "Peer" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid Event for Role", disdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid Event for Role", additional_detail := "null" }, StopTimer ResetCounters	CLOSED
R36	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "UCA_AckPDU" && Role = "Client" II "Peer" && GetCounterValue(UCC) > 0 && CIU = 0 DecrementCounter(UCC)	OPEN
R37	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "UCA_AckPDU" && Role = "Client" II "Peer" && GetCounterValue(UCC) > 0 && CIU > 0 DecrementCounter(UCC) , StartTimer(RC)	OPEN

#	Current State	Event Action	Next State
R38	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "UCA_AckPDU" && Role = "Client" II "Peer" && GetCounterValue(UCC) = 0 FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "UCA_AckPDU received and UCC=0", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "UCA_AckPDU received and UCC=0", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R39	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "IdlePDU" && CIU > 0 && Role = "Client" II "Peer" StartTimer(RC)	OPEN
R40	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "IdlePDU" && CIU > 0 && Role = "Server" II "Peer" StartTimer(RS)	OPEN
R41	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && FAL_Pdu_Type (dls_user_data) = "IdlePDU" && CIU = 0 FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED

#	Current State	Event Action	Next State
R42	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" && ((FAL_Pdu_Type (dls_user_data) <> "CS_RspPDU") (FAL_Pdu_Type (dls_user_data) <> "UCA_AckPDU") (FAL_Pdu_Type (dls_user_data) <> "IdlePDU")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R43	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Server" && ((FAL_Pdu_Type (dls_user_data) <> "CS_ReqPDU") (FAL_Pdu_Type (dls_user_data) <> "UCA_ReqPDU") (FAL_Pdu_Type (dls_user_data) <> "IdlePDU")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED

#	Current State	Event Action	Next State
R44	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Data_ind" && Role = "Client" && ((FAL_Pdu_Type (dls_user_data) <> "CS_ReqPDU") (FAL_Pdu_Type (dls_user_data) <> "CS_RspPDU") (FAL_Pdu_Type (dls_user_data) <> "UCA_ReqPDU") (FAL_Pdu_Type (dls_user_data) <> "UCA_AckPDU") (FAL_Pdu_Type (dls_user_data) <> "IdlePDU")) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R45	OPEN	RCTimer expired FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "RCTimer expired", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "RCTimer expired", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R46	OPEN	RSTimer expired FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "RSTimer expired", dlsdu := "null" }, Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "RSTimer expired", additional_detail := "null" }, StopTimer, ResetCounters	CLOSED

#	Current State	Event Action	Next State
R47	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (dls_user_data) = "Abort_PDU" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := AbortIdentifier (fal_pdu), reason_code := AbortReason (fal_pdu), additional_detail := AbortDetail (fal_pdu) }, StopTimer, ResetCounters	CLOSED
R48	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R49	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_user" Abort_ind{ arep_id := GetArepld (), locally_generated := "False", identifier := "FAL", reason_code := reason, additional_detail := "null" }, StopTimer, ResetCounters	CLOSED
R50	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" }, StopTimer, ResetCounters	CLOSED

#	Current State	Event Action	Next State
R51	NOT CLOSED	ErrorToARPM <pre> FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepId (), reason := reason, dlsdu := "null" }, Abort_ind { arep_id := GetArepId (), locally_generated := "True", identifier := "FAL", reason_code := reason, additional_detail := "null" }, StopTimer, ResetCounters </pre>	CLOSED

4.6.5.3.3 Functions used by QUB-FC ARPM

Table 93 – Function GetArepId ()

Name	GetArepId ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 94 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail		dlsdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 95 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data		One of the FAL-PDU types defined in the FAL-PDUs section.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FAL-PDU types.		

Table 96 – Function AREPContextCheck()

Name	AREPContextCheck()	Used in	ARPM																																																								
Input		Output																																																									
dl_sdu		Boolean value.																																																									
Function	This function checks the AREP context parameters that are received with establish service. The compatibility of the remote to the local context is shown in the following matrix:																																																										
<table><tr><th colspan="2"></th><th colspan="5">Local Context</th></tr><tr><th>Remote Context</th><th>Type</th><th>maxOSCC</th><th>maxOSCS</th><th>maxUCSC</th><th>maxUCSS</th><th>CI</th></tr><tr><td>Type</td><td>=</td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>maxOSCC</td><td></td><td></td><td></td><td>≥</td><td></td><td></td></tr><tr><td>mmaxOSCS</td><td></td><td>≤</td><td></td><td></td><td></td><td></td></tr><tr><td>maxUCSC</td><td></td><td></td><td></td><td></td><td>≥</td><td></td></tr><tr><td>maxUCSS</td><td></td><td></td><td>≤</td><td></td><td></td><td></td></tr><tr><td>CIU</td><td></td><td></td><td></td><td></td><td></td><td>=</td></tr></table>						Local Context					Remote Context	Type	maxOSCC	maxOSCS	maxUCSC	maxUCSS	CI	Type	=						maxOSCC				≥			mmaxOSCS		≤					maxUCSC					≥		maxUCSS			≤				CIU						=
		Local Context																																																									
Remote Context	Type	maxOSCC	maxOSCS	maxUCSC	maxUCSS	CI																																																					
Type	=																																																										
maxOSCC				≥																																																							
mmaxOSCS		≤																																																									
maxUCSC					≥																																																						
maxUCSS			≤																																																								
CIU						=																																																					
Explanation:																																																											
≤: local value smaller than or equal to remote value																																																											
≥: local value larger than or equal to remote value																																																											
=: local value equal to remote value																																																											

Table 97 – Function AbortIdentifier

Name	AbortIdentifier	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 98 – Function AbortReason

Name	AbortReason	Used in	ARPM
Input		Output	
fal_pdu		The Reason Code parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter		

Table 99 – Function AbortDetail

Name	AbortDetail	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.		

NOTE_The following two functions make use of persistent protocol timers that are able to issue the local events 'TSTimer expired', 'TCTimer expired', 'RCTimer expired' and 'RSTimer expired' to notify the expiration of the appropriate time interval to the ARPM.

Table 100 – Function StartTimer

Name	StartTimer	Used in	ARPM
Input		Output	
identifier			
Function	This function starts the selected persistent protocol timer as follows: If identifier is TS then function starts the TSTimer with the value of CIU/3. If identifier is TC then function starts the TCTimer with the value of CIU/3. If identifier is RS then function starts the RSTimer with the value of CIU. If identifier is RC then function starts the RCTimer with the value of CIU.		

NOTE_The appropriate timer is restarted if it was still running.

Table 101 – Function StopTimer

Name	StopTimer	Used in	ARPM
Input		Output	
Function	This function stops all local persistent protocol timers of the ARPM.		

NOTE_The following functions make use of local persistent variables OSCC (current value of outstanding services at client), UCC (current value of unconfirmed services at client) and OSCS (current value of outstanding services at server) that supports counting of outstanding services. The initial values of OSCC, UCC, OSCS are 0.

Table 102 – Function ResetCounters

Name	ResetCounters	Used in	ARPM
Input		Output	
Function	This function sets OSCC, UCC, OSCS to zero.		

Table 103 – Function IncrementCounter

Name	IncrementCounter	Used in	ARPM
Input		Output	
identifier			
Function	This function increments the selected counter.		

Table 104 – Function DecrementCounter

Name	DecrementCounter	Used in	ARPM
Input		Output	
identifier			
Function	This function decrements the selected counter.		

Table 105 – Function GetCounterValue

Name	GetCounterValue	Used in	ARPM
Input		Output	
identifier		value	
Function	This function returns the current value of the selected counter.		

NOTE_The following function makes use of local persistent variable XON_OFF_Flag that stores the current value of flow control from the AR_XON_OFF_ReqPDU. The initial value of XON_OFF_Flag is True.

Table 106 – Function Xon

Name	Xon	Used in	ARPM
Input		Output	
(none)		Boolean (True/False)	
Function	This function returns True if XON_OFF_Flag was True otherwise it returns False.		

Table 107 – Function SetXonOff

Name	SetXonOff	Used in	ARPM
Input		Output	
user_data			
Function	This function extracts the XON_OFF octet from user_data. It sets XON_XOFF_Flag True if XON_OFF is 'ON' within the PDU otherwise it sets the flag False.		

4.6.6 Buffered Usertriggered Bidirectional (BUB) ARPM

4.6.6.1 Primitive Definitions

4.6.6.1.1 Primitives Exchanged between ARPM and FSPM

Table 108 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
UCS_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.
FCMP_req	FSPM	(none)	This is an FAL internal primitive used to convey a FAL-Compel (FCMP) request primitive from the FSPM to the ARPM.
GBM_req	FSPM	(none)	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) request primitive from the FSPM to the ARPM.

Table 109 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated_identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, remote_dlsap_address, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.
FCMP_cnf	ARPM	arep_id, status	This is an FAL internal primitive used to convey a FAL-Compel (FCMP) confirmation primitive from the ARPM to the FSPM.
GBM_cnf(+)	ARPM	arep_id, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) positive confirmation primitive from the ARPM to the FSPM.
GBM_cnf(-)	ARPM	arep_id	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) negative confirmation primitive from the ARPM to the FSPM.
FSTS_ind	ARPM	arep_id, reported_status	This is an FAL internal primitive used to convey a FAL-Status (FSTS) indication primitive from the ARPM to the FSPM.

4.6.6.1.1.1 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 110.

Table 110 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL-SDU parameter.
remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Link Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.6.2 DLL Mapping of BUB AREP Class

This clause describes the mapping of the BUB AREP class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data Link Layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the BUB AREP class are defined in this clause.

CLASS: BubCo

PARENT CLASS: BufferedUser-TriggeredBidirectionalAREP

ATTRIBUTES:

- 1 (m) KeyAttribute: LocalDlcepAddress
- 2 (m) Attribute: RemoteDlcepAddress
- 3 (m) Attribute: DlsapRole (Initiator, Constrained)
- 4 (m) Attribute: QosParameterSet
- 4.1 (m) Attribute: DlcepClass (Peer)
- 4.2 (m) Attribute: DlcepDataDeliveryFeatures
- 4.2.1 (m) Attribute: FromRequestorToResponder (Ordered, Unordered)
- 4.2.2 (m) Attribute: FromResponderToRequestor (Ordered, Unordered)
- 4.3 (m) Attribute: Priority
- 4.3.1 (m) Attribute: DllPriority (Urgent, Normal, TimeAvailable)
- 4.3.2 (m) Attribute: DllPriorityNegotiated (Urgent, Normal, TimeAvailable)
- 4.4 (m) Attribute: Dl pduAuthentication (Ordinary, Source, Maximal)
- 4.5 (m) Attribute: ResidualActivity
- 4.5.1 (m) Attribute: ResidualActivityAsSender (True, False)
- 4.5.2 (m) Attribute: ResidualActivityAsReceiver (True, False)
- 4.6 (m) Attribute: MaxConfirmDelay
- 4.7 (c) Constraint: DlsapRole = Initiator
- 4.7.1 (m) Attribute: MaxConfirmDelayOnDIConnect
- 4.7.2 (m) Attribute: MaxConfirmDelayOnDIData
- 4.8 (m) Attribute: DISchedulingPolicy (Explicit)
- 4.9 (m) Attribute: MaxDlsduSizes
- 4.9.1 (m) Attribute: MaxDlsduSizeFromRequestor
- 4.9.2 (m) Attribute: MaxDlsduSizeFromResponder
- 4.9.3 (m) Attribute: MaxDlsduSizeFromRequestorNegotiated
- 4.9.4 (m) Attribute: MaxDlsduSizeFromResponderNegotiated

DLL SERVICES:

- 1 (m) OpsService: DL-Put
- 2 (m) OpsService: DL-Get
- 3 (m) OpsService: DL-Connect
- 4 (m) OpsService: DL-Connection-Established
- 5 (m) OpsService: DL-Disconnect
- 6 (m) OpsService: DL-Buffer-Received
- 7 (m) OpsService: DL-Buffer-Sent
- 8 (m) OpsService: DL-Compel-Service

4.6.6.2.1 Attributes

4.6.6.2.1.1 LocalDlcepAddress

This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the “DLCEP-address” parameter of the DLL.

NOTE1 The value of this attribute is also carried in the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

4.6.6.2.1.2 RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP.

This attribute supplies the value for called DLCEP-address of the DL-Connect service.

NOTE1 The value of this attribute is also carried in the header part of the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

4.6.6.2.1.3 DisapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP.

This attribute supplies the value for the “DL(SAP)-role” parameter. The permitted values Initiator, Constrained Responder and Unconstrained Responder, correspond respectively to INITIATOR, CONSTRAINED, and UNCONSTRAINED as defined in the Data Link Layer standard.

4.6.6.2.1.4 QosParameterSet

The QosParameterSet attributes specify the DL quality of service that is used by the AREP. These attribute values may be negotiated with the remote AREP.

4.6.6.2.1.5 DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP.

This attribute supplies the value for the “DLCEP class” parameter of the DLL. The possible value of this attribute is Peer and corresponds to PEER defined by the DLL.

4.6.6.2.1.6 DlcepDataDeliveryFeatures

These two attributes specify the data delivery features of the DLL.

The permitted values Ordered and Unordered correspond respectively to ORDERED and UNORDERED defined by the Data Link Layer standard.

The FromRequestorToResponder and FromResponderToRequestor attributes shall have the same value.

4.6.6.2.1.7 FromRequestorToResponder

This attribute specifies the DLL data delivery feature of the AREP as a sender. It supplies the value for the “DLCEP data delivery features from requestor to responder(s)” parameter defined in the DLL.

4.6.6.2.1.8 FromResponderToRequestor

This attribute specifies the DLL data delivery feature of the AREP as a receiver. It supplies the value for the “DLCEP data delivery features from responder(s) to requestor” parameter defined in the DLL.

4.6.6.2.1.9 Priority

4.6.6.2.1.10 DIIPriority

This attribute specifies the DLL priority before negotiation.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same BUB AREP. Also it is not permitted to use different priorities for the send and the receive conveyance paths.

4.6.6.2.1.11 DllPriorityNegotiated

This attribute specifies the DLL priority after negotiation.

4.6.6.2.1.12 DlpduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the “DLPDU-authentication” parameter of the DLL. The permitted value Ordinary, Source and Maximal correspond to ORDINARY, SOURCE and MAXIMAL respectively, as defined in the Fieldbus Data Link Layer standard.

4.6.6.2.1.13 ResidualActivityAsSender

This attribute specifies sender’s DLC residual activity. It supplies the value for the “Residual activity as sender” parameter as defined in the DLL. The possible values are “True” and “False”.

4.6.6.2.1.14 ResidualActivityAsReceiver

This attribute specifies receiver’s DLC residual activity. It supplies the value for the “Residual activity as receiver” parameter defined in the DLL. The possible values are “True” and “False”.

4.6.6.2.1.15 MaxConfirmDelay

This attribute specifies the maximum confirmation delay of certain DLL connection-oriented services.

4.6.6.2.1.16 MaxConfirmDelayOnDIConnect

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query” parameter specified in the Data Link Layer standard.

4.6.6.2.1.17 MaxConfirmDelayOnDIData

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Data service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Data” parameter specified in the Fieldbus Foundation Data Link Layer standard.

4.6.6.2.1.18 DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as possible.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Explicit corresponds to EXPLICIT defined in the Fieldbus Data Link Layer standard.

4.6.6.2.1.19 MaxDlsduSizeFromRequestor

This attribute specifies the maximum length of an FAL-PDU that can be sent from this AREP before negotiation.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

4.6.6.2.1.20 MaxDlsduSizeFromResponder

This attribute specifies the maximum length of an FAL-PDU that can be received by this AREP before negotiation.

This attribute supplies the value for the “Maximum DLSDU sizes from responder” parameter of the DLL.

4.6.6.2.1.21 MaxDlsduSizeFromRequestorNegotiated

This attribute specifies the maximum length of an FAL-PDU that can be sent from this AREP after negotiation.

4.6.6.2.1.22 MaxDlsduSizeFromResponderNegotiated

This attribute specifies the maximum length of an FAL-PDU that can be received by this AREP after negotiation.

4.6.6.3 BUB ARPM State Machine

4.6.6.3.1 BUB ARPM States

Table 111 - BUB ARPM States

CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs. It may send or receive Establish service FAL-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
REQUESTING (REQ)	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING (RSP)	The AREP has received an Establish Request FAL-PDU, delivered an Establish.indication primitive and is waiting for a response from its user.
REPLIED (REPL)	The Server AREP has issued an EST_rsp(+) primitive and is waiting for a "connection established" indication from the DLL.
SAME	Indicates that the next state is the same as the current state.

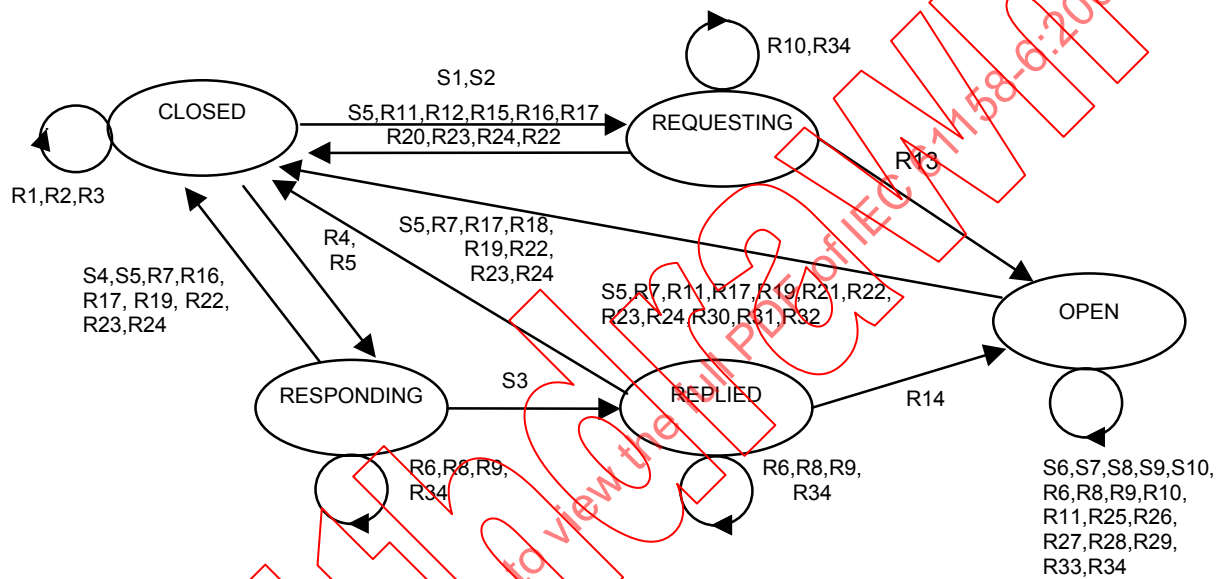


Figure 31 – State Transition Diagram of BUB ARPM

4.6.6.3.2 BUB ARPM State Table

Table 112 - BUB ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType:= "Free" RemoteDlcepAddress:= remote_dlcep_address, FAL-PDU_req { dmpm_service_name:= "DMPM_Connect_req", arep_id:= GetArepld(), called_address:= "default dlcep address", calling_address:= "default dlcep address", local_dlcep_address:= LocalDlcep, dlcsdu:= BuildFAL-PDU (fal_pdu_name:= "EST_ReqPDU", calling_dlcep_address:= LocalDlcepAddress, called_dlcep_address:= RemoteDlcepAddress, fal_data:= user_data) }	REQ

#	Current State	Event Action	Next State
S2	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType:= "Linked" FAL-PDU_req { dmpm_service_name:= "DMPM_Connect_req", arep_id:= GetArepld(), called_address:= "default dlsap address", calling_address:= "default dlsap address", local_dlcep_address:= LocalDlcep, dlsdu:= BuildFAL-PDU (fal_pdu_name:= "EST_ReqPDU", calling_dlcep_address:= LocalDlcepAddress, called_dlcep_address:= RemoteDlcepAddress, fal_data:= user_data) }	REQ
S3	RSP	EST_rsp(+) FAL-PDU_req { dmpm_service_name:= "DMPM_Connect_rsp", arep_id:= GetArepld(), responding_address:= "default dlsap address", local_dlcep_address:= LocalDlcep, dlcep_dl_id:= DlcepDlIdentifier, dlsdu:= BuildFAL-PDU (fal_pdu_name:= "EST_RspPDU", fal_data:= user_data) }	REPL
S4	RSP	EST_rsp(-) FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "connection rejection-transient condition", dlsdu:= BuildFAL-PDU (fal_pdu_name:= "EST_ErrPDU", fal_data:= user_data) }	CLOSED
S5	NOT CLOSED	Abort_req FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "disconnection-normal condition", dlsdu:= BuildFAL-PDU (fal_pdu_name:= "Abort_PDU", fal_id:= identifier, fal_reason_code:= reason_code, fal_additional_detail:= additional_detail) }	CLOSED
S6	OPEN	CS_req && Role = "Client" "Peer" FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "CS_ReqPDU", fal_data:= user_data) }	OPEN
S7	OPEN	CS_rsp && Role = "Server" "Peer" FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "CS_RspPDU", fal_data:= user_data) }	OPEN

#	Current State	Event Action	Next State
S8	OPEN	FCMP_req && Role = "Client" "Server" "Peer" FAL-PDU_req { dmpm_service_name:= "DMPM_Compel_Service_req", arep_id:= GetArepld(), action_class:= "Local" }	OPEN
S9	OPEN	GBM_req && Role = "Client" "Server" "Peer" FAL-PDU_req { dmpm_service_name:= "DMPM_Get_req", arep_id:= GetArepld() }	OPEN
S10	OPEN	UCS_req FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "UCS_ReqPDU", fal_data:= user_data) }	OPEN

Table 113 - BUB state table - Receiver transactions

#	Current State	Event Action	Next State
R1	CLOSED	Connect_ind && Initiator = "True" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Multiple Initiators", dlsdu:= "null" }	CLOSED
R2	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) <> "EST_ReqPDU" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Invalid FAL-PDU", dlsdu:= "null" }	CLOSED
R3	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Remote Address Mismatch", dlsdu:= "null" }	CLOSED

#	Current State	Event Action	Next State
R4	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" RemoteDlcepAddress:= calling_dlcep_address, DlcepDlIdentifier:= dlcep_dl_id, MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated: = dlsdu_size_from_responder, EST_ind { arep_id:= GetArepld (), user_data:= dls_user_data }	RSP
R5	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress = calling_dlcep_address DlcepDlIdentifier:= dlcep_dl_id, MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, EST_ind { arep_id:= GetArepld (), user_data:= dls_user_data }	RSP
R6	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Remote Address Mismatch", dlsdu:= "null" }	SAME
R7	RSP REPL OPEN	Connect_ind && Initiator = "False" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" }	CLOSED
R8	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "AREP Busy", dlsdu:= "null" }	SAME

#	Current State	Event Action	Next State
R9	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) <> "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Invalid FAL-PDU", dlsdu:= "null" } 	SAME
R10	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Multiple Initiators", dlsdu:= "null" } 	SAME
R11	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= dlcep_dl_id (from Connect_ind), reason:= "Multiple Initiators", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Multiple Initiators" } 	CLOSED
R12	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) <> "EST-RspPDU" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDllIdentifier, reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" } 	CLOSED
R13	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) = "EST-RspPDU" MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, DllPriorityNegotiated:= dll_priority, EST_cnf(+) { arep_id:= GetArepld (), user_data:= dls_user_data } 	OPEN

#	Current State	Event Action	Next State
R14	REPL	FAL-PDU_ind && dmpm_service_name = "DMPM_Connection_Established_ind" (no actions taken)	OPEN
R15	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && FAL_Pdu_Type (fal_pdu) = "EST-ErrPDU" EST_cnf(-) { arep_id:= GetArepld (), user_data:= dls_user_data }	CLOSED
R16	REQ RSP	FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "Invalid DI PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid DI PDU" }	CLOSED
R17	NOT CLOSED	FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (fal_pdu) = "Abort_PDU" Abort_ind { arep_id:= GetArepld (), locally_generated:= "False", identifier:= AbortIdentifier (fal_pdu), reason_code:= AbortReason (fal_pdu), additional_detail:= AbortDetail (fal_pdu) }	CLOSED
R18	REPL	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DM_Connection_Established_ind")) FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "Invalid DI PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid DI PDU" additional_detail:= "null" }	CLOSED

#	Current State	Event Action	Next State
R19	REPL RSP OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (fal_pdu) <> "Abort_PDU" Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" }	CLOSED
R20	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && ((FAL_Pdu_Type (fal_pdu) <> "Abort_PDU") && (FAL_Pdu_Type (fal_pdu) <> "EST_ErrPDU") Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" }	CLOSED
R21	OPEN	FAL-PDU_ind && ((FAL_Pdu_Type (fal_pdu) <> "DMPM_Disconnect_ind") && (FAL_Pdu_Type (fal_pdu) <> "DMPM_Put_Out") FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDllIdentifier, reason:= "Invalid DI PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid DI PDU" additional_detail:= "null" }	CLOSED
R22	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" Abort_ind { arep_id:= GetArepld (), locally_generated:= "False", identifier:= "Data Link Layer", reason_code:= reason additional_detail:= "null" }	CLOSED
R23	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_user" Abort_ind { arep_id:= GetArepld (), locally_generated:= "False", identifier:= "FAL", reason_code:= reason additional_detail:= "null" }	CLOSED

#	Current State	Event Action	Next State
R24	NOT CLOSED	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "Data Link Layer", reason_code:= reason additional_detail:= "null" } </pre>	CLOSED
R25	OPEN	<pre> FAL-PDU_ind && Role = "Peer" "Server" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) = "CS_ReqPDU" CS_ind { arep_id:= GetArepld (), user_data:= fal_pdu } </pre>	OPEN
R26	OPEN	<pre> FAL-PDU_ind && Role = "Client" "Peer" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) = "CS_RspPDU" CS_cnf { Arep_id:= GetArepld (), user_data:= fal_pdu } </pre>	OPEN
R27	OPEN	<pre> FAL-PDU_ind && Role = "Peer" "Client" "Server" && dmpm_service_name = "DMPM_Buffer_Sent_ind" FSTS_ind { arep_id:= GetArepld(), reported_status:= "Buffer_Sent" } </pre>	OPEN
R28	OPEN	<pre> FAL-PDU_ind && Role = "Peer" "Client" "Server" && dmpm_service_name = "DMPM_Get_cnf" && status = "success" GMB_cnf(+){ arep_id:= GetArepld(), duplicate_fal_sdu=duplicate_dlsdu, user_data:=fal_pdu } </pre>	OPEN
R29	OPEN	<pre> FAL-PDU_ind && Role = "Peer" "Client" "Server" && dmpm_service_name = "DMPM_Get_cnf" && status <> "success" GMB_cnf(-){ arep_id:= GetArepld() } </pre>	OPEN

#	Current State	Event Action	Next State
R30	OPEN	<pre> FAL-PDU_ind && Role = "Server" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) <> "CS_ReqPDU" && FAL_Pdu_Type (fal_pdu) <> "UCS_ReqPDU" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" } </pre>	CLOSED
R31	OPEN	<pre> FAL-PDU_ind && Role = "Client" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU" && FAL_Pdu_Type (fal_pdu) <> "UCS_ReqPDU" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True" identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" } </pre>	CLOSED
R32	OPEN	<pre> FAL-PDU_ind && Role = "Peer" && dmpm_service_name = "DMPM_Buffer_Received_ind" && ((FAL_Pdu_Type (fal_pdu) <> "CS_ReqPDU") && (FAL_Pdu_Type (fal_pdu) <> "CS_RspPDU") && (FAL_Pdu_Type (fal_pdu) <> "UCS_ReqPDU")) FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), dlcep_dl_id:= DlcepDlIdentifier, reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" } </pre>	CLOSED

#	Current State	Event Action	Next State
R33	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) = "UCS_ReqPDU" UCS_ind { arep_id:= GetArepld (), user_data:= fal_pdu }	OPEN
R34	NOT CLOSED	ErrorToARPM (no action taken) NOTE It is a local matter to report this error status to network management entities. The ARPM does not abort the existing connections. The FAL user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive.	SAME

4.6.6.3.3 Functions used by BUB ARPM

Table 114 – Function GetArepld ()

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 115 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
	fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail	Disdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 116 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data		One of the FAL-PDU types defined in the FAL-PDUs section.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FAL-PDU types.		

Table 117 – Function AbortIdentifier

Name	AbortIdentifier	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 118 – Function AbortReason

Name	AbortReason	Used in	ARPM
Input		Output	
fal_pdu		The Reason Code parameter of the Abort service.	
Function			
This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter.			

Table 119 – Function AbortDetail

Name	AbortDetail	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function			
This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.			

4.6.7 Buffered Networkscheduled Bidirectional (BNB) ARPM

4.6.7.1 Primitive Definitions

4.6.7.1.1 Primitives Exchanged between ARPM and FSPM

Table 120 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
UCS_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 121 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, remote_dlsap_address, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.

4.6.7.2 Primitives issued by ARPM to FSPMParameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 122.

Table 122 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL-SDU parameter.
remote_dlsap_address	This parameter conveys value that is used for the Remote_DLSAP_Address parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Link Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.7.3 DLL Mapping of BNB AREP Class

This clause describes the mapping of the BNB AREP Class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data Link Layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the BNB AREP class are defined in this clause.

CLASS: Bnb

PARENT CLASS: BufferedNetwork-TriggeredBidirectionalAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalDlcepAddress
2	(m)	Attribute:	RemoteDlcepAddress
3	(m)	Attribute:	DlsapRole (Basic)
4	(m)	Attribute:	QosParameterSet
4.1	(m)	Attribute:	DlcepClass (Peer)
4.2	(m)	Attribute:	DlcepDataDeliveryFeatures
4.2.1	(m)	Attribute:	FromRequestorToResponder (Ordered)
4.2.2	(m)	Attribute:	FromResponderToRequestor (Ordered)
4.3	(m)	Attribute:	Priority
4.3.1	(m)	Attribute:	DllPriority (Urgent, Normal, TimeAvailable)
4.3.2	(m)	Attribute:	DllPriorityNegotiated (Urgent, Normal, TimeAvailable)
4.4	(m)	Attribute:	DlpduAuthentication (Ordinary, Source, Maximal)
4.5	(m)	Attribute:	ResidualActivity
4.5.1	(m)	Attribute:	ResidualActivityAsSender (True, False)
4.5.2	(m)	Attribute:	ResidualActivityAsReceiver (True, False)
4.6	(m)	Attribute:	MaxConfirmDelay
4.6.1	(m)	Attribute:	MaxConfirmDelayOnDlConnect
4.6.2	(m)	Attribute:	MaxConfirmDelayOnDlData
4.7	(m)	Attribute:	DlSchedulingPolicy (Explicit)
4.8	(m)	Attribute:	MaxDisduSizes
4.8.1	(m)	Attribute:	MaxDisduSizeFromRequestor
4.8.2	(m)	Attribute:	MaxDisduSizeFromResponder
4.8.3	(m)	Attribute:	MaxDisduSizeFromRequestorNegotiated
4.8.4	(m)	Attribute:	MaxDisduSizeFromResponderNegotiated

DLL SERVICES:

1	(m)	OpsService:	DL-Put
2	(m)	OpsService:	DL-Get
3	(m)	OpsService:	DL-Connect
4	(m)	OpsService:	DL-Connection-Established
5	(m)	OpsService:	DL-Disconnect
6	(m)	OpsService:	DL-Buffer-Received
7	(m)	OpsService:	DL-Buffer-Sent

4.6.7.3.1 Attributes

4.6.7.3.1.1 LocalDlcepAddress

This attribute specifies the local DLCEP address and identifies the DLCEP. The value of this attribute is used as the "DLCEP-address" parameter of the DLL.

NOTE1 The value of this attribute is also carried in the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

4.6.7.3.1.2 RemoteDlcepAddress

This attribute specifies the remote DLCEP address and identifies the DLCEP.

This attribute supplies the value for called DLCEP-address of the DL-Connect service.

NOTE1 The value of this attribute is also carried in the header part of the Establish Request PDU.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE2 Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

4.6.7.3.1.3 DisapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP.

This attribute supplies the value for the “DL(SAP)-role” parameter.

4.6.7.3.1.4 QosParameterSet

The QosParameterSet attributes specify the DL quality of service that is used by the AREP. These attribute values may be negotiated with the remote AREP.

DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP.

This attribute supplies the value for the “DLCEP class” parameter of the DLL. The possible value of this attribute is Peer and corresponds to PEER defined by the DLL.

DlcepDataDeliveryFeatures

These attributes specify the data delivery features of the DLL.

The permitted value Ordered corresponds to ORDERED defined by the Data Link Layer standard.

NOTE The FromRequestorToResponder and FromResponderToRequestor attributes shall have the same value.

FromRequestorToResponder

This attribute specifies the DLL data delivery feature of the AREP as a sender. It supplies the value for the “DLCEP data delivery features from requestor to responder(s)” parameter defined in the DLL.

FromResponderToRequestor

This attribute specifies the DLL data delivery feature of the AREP as a receiver. It supplies the value for the “DLCEP data delivery features from responder(s) to requestor” parameter defined in the DLL.

Priority

DllPriority

This attribute specifies the DLL priority before negotiation.

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same BNB AREP. Also, it is not permitted to use different priorities for the send and the receive conveyance paths.

DllPriorityNegotiated

This attribute specifies the DLL priority after negotiation.

DlpduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the “DLPDU-authentication” parameter of the DLL. The permitted value Ordinary, Source and Maximal correspond to ORDINARY, SOURCE and MAXIMAL respectively, as defined in the Fieldbus Data Link Layer standard.

ResidualActivityAsSender

This attribute specifies sender’s DLC residual activity. It supplies the value for the “Residual activity as sender” parameter as defined in the DLL. The possible values are “True” and “False”.

ResidualActivityAsReceiver

This attribute specifies receiver’s DLC residual activity. It supplies the value for the “Residual activity as receiver” parameter defined in the DLL. The possible values are “True” and “False”.

MaxConfirmDelay

This attribute specifies the maximum confirmation delay of certain DLL connection-oriented services.

MaxConfirmDelayOnDIConnect

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query” parameter specified in the Data Link Layer standard.

MaxConfirmDelayOnDIData

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Data service.

This attribute supplies the value for the “Maximum confirmation delay on DL-Data” parameter specified in the Data Link Layer standard (IEC 61158-3).

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU as soon as possible.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Explicit corresponds to EXPLICIT defined in the Fieldbus Data Link Layer standard.

MaxDisduSize***MaxDisduSizeFromRequestor***

This attribute specifies the maximum length of an FAL-PDU that can be sent from this AREP before negotiation.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

MaxDisduSizeFromResponder

This attribute specifies the maximum length of an FAL-PDU that can be received by this AREP before negotiation.

This attribute supplies the value for the “Maximum DLSDU sizes from responder” parameter of the DLL.

MaxDisduSizeFromRequestorNegotiated

This attribute specifies the maximum length of an FAL-PDU that can be sent from this AREP after negotiation.

MaxDisduSizeFromResponderNegotiated

This attribute specifies the maximum length of an FAL-PDU that can be received by this AREP after negotiation.

4.6.7.3.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.7.4 BNB ARPM state machine

4.6.7.4.1 BNB ARPM States

Table 123 – BNB ARPM States

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs. It may send or receive Establish service FAL-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
OPEN_CSC	The AREP is defined and capable of sending Confirmed Request FAL-PDUs as client.
OPEN_CSS	The AREP is defined and capable of sending Confirmed Response FAL-PDUs as server.
OPEN_UCSC	The AREP is defined and capable of sending Unconfirmed Request FAL-PDUs as client.
REQUESTING (REQ)	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING (RSP)	The AREP has received an Establish Request FAL-PDU, delivered an Establish indication primitive and is waiting for a response from its user.
REPLIED (REPL)	The Server AREP has issued an EST_rsp(+) primitive and is waiting for receiving a "connection established" indication from the DLL.
WAIT_SENT_CSC	The AREP has put a Confirmed Request FAL-PDU into the buffer and is waiting for a "DMPM_Buffer_Sent" indication from the DLL.
WAIT_SENT_CSS	The AREP has put a Confirmed Response FAL-PDU into the buffer and is waiting for a "DMPM_Buffer_Sent" indication from the DLL.
WAIT_SENT_UCSC	The AREP has put an Unconfirmed Request FAL-PDU into the buffer as client and is waiting for a "DMPM_Buffer_Sent" indication from the DLL.
WAIT_RESOURCE_CSS	The AREP is waiting for a free resource (buffer) to send a Confirmed Response FAL-PDU.
WAIT_RESOURCE_CSC	The AREP is waiting for a free resource (buffer) to send a Confirmed Request FAL-PDU.
WAIT_CYC_RESOURCE	The AREP is waiting for a free resource (buffer) to send a Confirmed Request FAL-PDU.
WAIT_RESOURCE_UCSC	The AREP is waiting for a free resource (buffer) to send an Unconfirmed Request FAL-PDU.
WAIT_RES	The AREP is capable of receiving a Confirmed Response FAL-PDU.
WAIT_CON	The AREP is capable of getting a Confirmed Response FAL-PDU from the buffer.
WAIT_CYC_RES	The AREP is capable of receiving a Confirmed Response FAL-PDU and sending a Confirmed Request FAL-PDU cyclic.
WAIT_CYC_IND	The AREP is capable of receiving a Confirmed Request FAL-PDU and sending a Confirmed Response FAL-PDU cyclic.
WAITING_CS	The AREP is waiting for a Confirmed Response FAL-PDU after receiving an Unconfirmed Request FAL-PDU.
SAME	Indicates that the next state is the same as the current state.

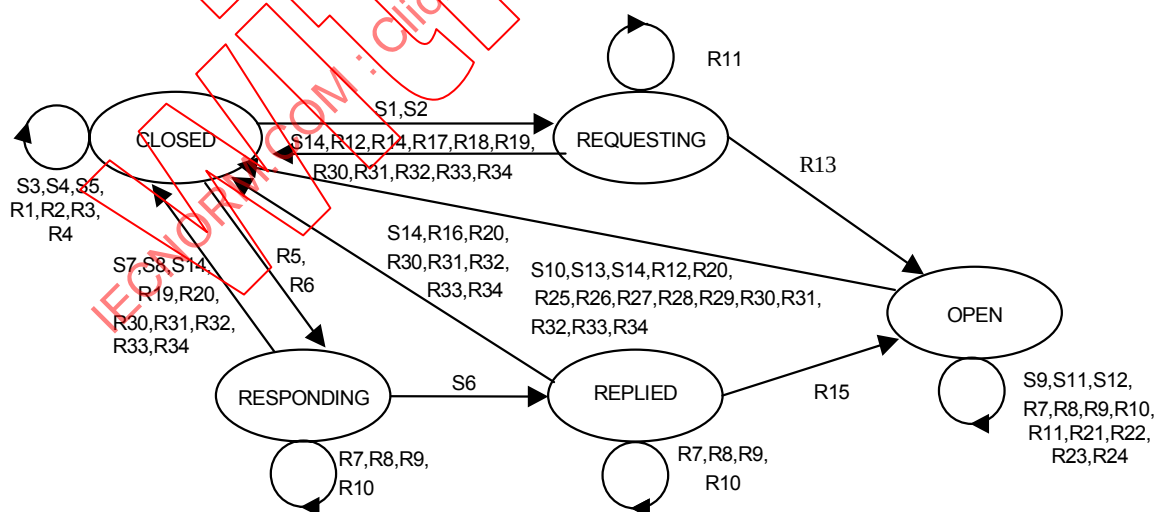


Figure 32 – State Transition Diagram of BNB ARPM (Basic State Machine)

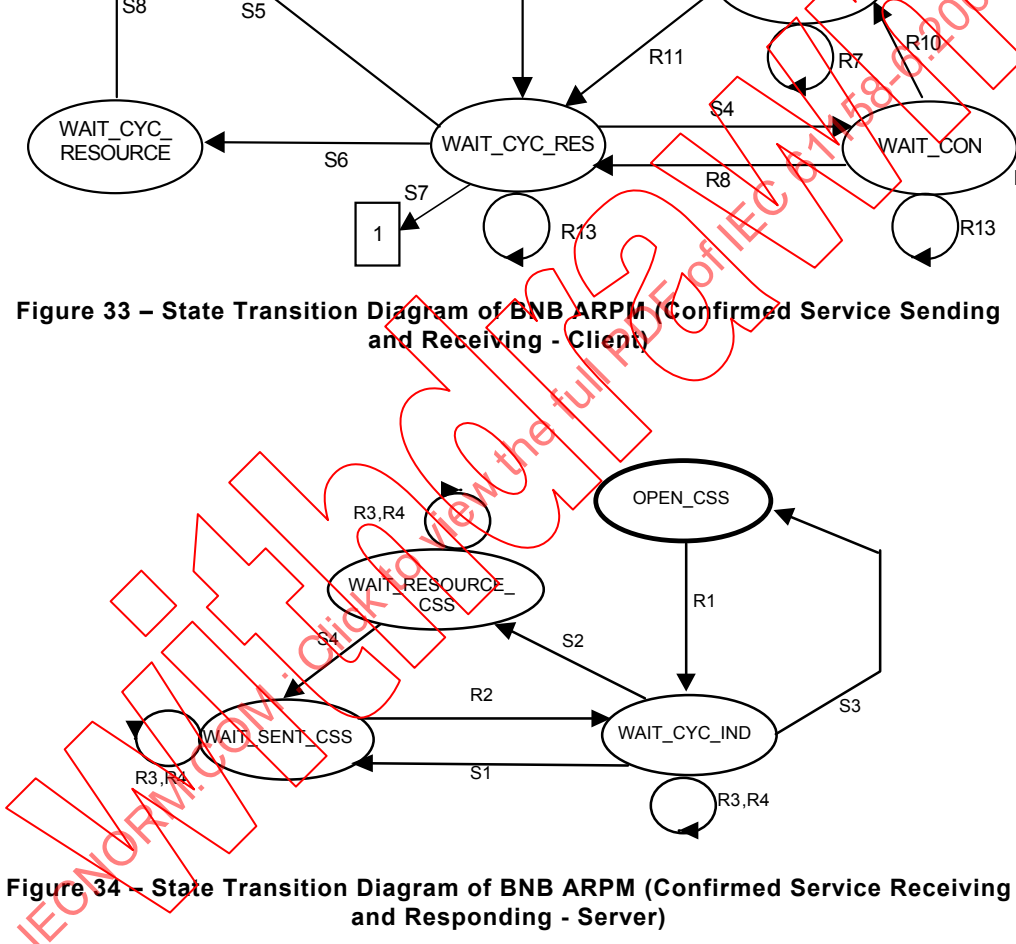


Figure 33 – State Transition Diagram of BNB ARPM (Confirmed Service Sending and Receiving - Client)

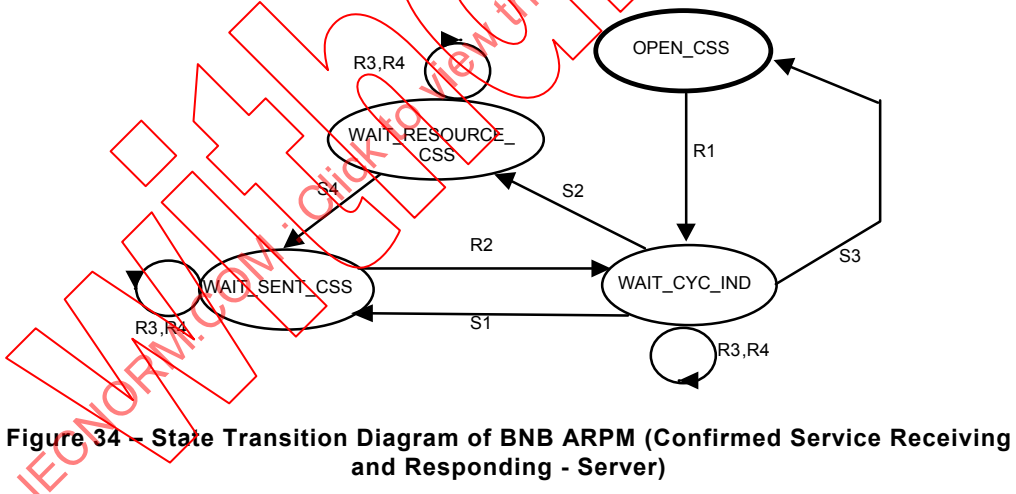


Figure 34 – State Transition Diagram of BNB ARPM (Confirmed Service Receiving and Responding - Server)

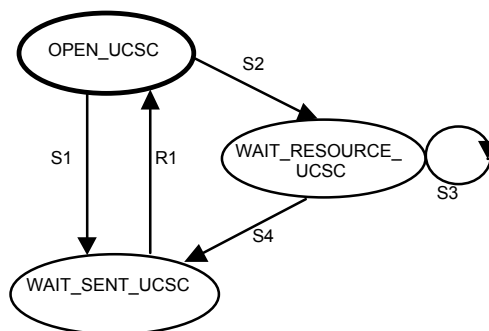


Figure 35 – State Transition Diagram of BNB ARPM (Unconfirmed Service Sending- Client)

4.6.7.4.2 BNB ARPM State Table (Basic State Machine)

Table 124 – BNB ARPM State Table - Sender Transactions (Basic State Machine)

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType = "Free" RemoteDlcepAddress := remote_dlcep_address, FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) }	REQ
S2	CLOSED	EST_req && Initiator = "True" && RemoteAddressConfigurationType = "Linked" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "default dlsap address", calling_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ReqPDU", calling_dlcep_address := LocalDlcepAddress, called_dlcep_address := RemoteDlcepAddress, fal_data := user_data) }	REQ
S3	CLOSED	EST_req && Initiator = "False" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid Event for Role" }	CLOSED
S4	CLOSED	unallowed primitive Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed AREP primitive in connection establishment" }	CLOSED

#	Current State	Event Action	Next State
S5	CLOSED	Abort_req (ignore)	CLOSED
S6	RSP	EST_rsp(+) FAL-PDU_req { dmpm_service_name := "DMPM_Connect_rsp", arep_id := GetArepld (), responding_address := "default dlsap address", local_dlcep_address := LocalDlcep, dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_RspPDU", fal_data := user_data) } }	REPL
S7	RSP	EST_rsp(-) FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "connection rejection-transient condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "EST_ErrPDU", fal_data := user_data) } }	CLOSED
S8	RSP	unallowed primitive Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed AREP primitive in connection establishment " }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := " Unallowed AREP primitive in connection ", dlsdu := "null" } }	CLOSED
S9	OPEN	CS_req && VA_ASE_Service = "Read.req" && status CS Req. = OPEN_UCSC start CS Req.	OPEN
S10	OPEN	CS_req && VA_ASE_Service <> "Read.req" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed AREP primitive " }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := " Unallowed AREP primitive ", dlsdu := "null" } }, reset all machines, StopTimer	CLOSED
S11	OPEN	UCS_req && status UCSC Req. = OPEN_UCSC start UCSC Req.	OPEN
S12	OPEN	UCS_req && status UCSC Req. <> OPEN_UCSC (no actions taken)	OPEN

#	Current State	Event Action	Next State
S13	OPEN	unallowed Services Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Unallowed AREP primitive " }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Error of Remote Station", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) }, reset all machines, StopTimer	CLOSED
S14	NOT CLOSED	Abort_req FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) }, reset all machines, StopTimer	CLOSED

Table 125 – BNB State Table - Receiver Transactions (Basic State Machine)

#	Current State	Event Action	Next State
R1	CLOSED	Connect_ind && Initiator = "True" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Multiple Initiators", dlsdu:= "null"	CLOSED
R2	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) <> "EST_ReqPDU" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid FAL-PDU", dlsdu:= "null" }	CLOSED

#	Current State	Event Action	Next State
R3	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Remote Address Mismatch", dlsdu:= "null" }	CLOSED
R4	CLOSED	unallowed primitive (no actions taken)	CLOSED
R5	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress = calling_dlcep_address MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder EST_ind { arep_id:= GetArepld (), user_data:= dls_user_data }	RSP
R6	CLOSED	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" RemoteDlcepAddress:= calling_dlcep_address, MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated:= dlsdu_size_from_responder, EST_ind { arep_id:= GetArepld (), user_data:= dls_user_data }	RSP
R7	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Linked" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Remote Address Mismatch", dlsdu:= "null" }	SAME

#	Current State	Event Action	Next State
R8	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" } reset all machines, StopTimer	CLOSED
R9	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) = "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "AREP Busy", dlsdu:= "null" }	SAME
R10	RSP REPL OPEN	Connect_ind && Initiator = "False" && FAL_Pdu_type (dls_user_data) <> "EST_ReqPDU" && RemoteAddressConfigurationType = "Free" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid FAL-PDU", dlsdu:= "null" }	SAME
R11	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress <> calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Multiple Initiators", dlsdu:= "null" }	SAME

#	Current State	Event Action	Next State
R12	REQ OPEN	Connect_ind && Initiator = "True" && RemoteDlcepAddress = calling_dlcep_address FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Multiple Initiators", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Multiple Initiators" } reset all machines, StopTimer	CLOSED
R13	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) = "EST-RspPDU" MaxDlsduSizeFromRequestorNegotiated := dlsdu_size_from_requestor, MaxDlsduSizeFromResponderNegotiated := dlsdu_size_from_responder, DlIPriorityNegotiated:= dll_priority, EST_cnf(+) { arep_id:= GetArepld (), user_data:= dls_user_data }	OPEN
R14	REQ	Connect_cnf && FAL_Pdu_Type (dls_user_data) <> "EST-RspPDU" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" }	CLOSED
R15	REPL	FAL-PDU_ind && dmpm_service_name = "DMPM_Connection_Established_ind" (no actions taken)	OPEN
R16	REPL	FAL-PDU_ind && ((dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DM_Connection_Established_ind")) FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid DI PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid DI PDU" additional_detail:= "null" }	CLOSED

#	Current State	Event Action	Next State
R17	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && FAL_Pdu_Type (dls_user_data) = "EST-ErrPDU" EST_cnf(-) { arep_id:= GetArepld (), user_data:= dls_user_data }	CLOSED
R18	REQ	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && ((FAL_Pdu_Type (dls_user_data) <> "Abort_PDU") && (FAL_Pdu_Type (dls_user_data) <> "EST_ErrPDU") Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" }	CLOSED
R19	REQ RSP	FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid DI PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid DI PDU" }	CLOSED
R20	REPL RSP OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (dls_user_data) <> "Abort_PDU" Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" } reset all machines, StopTimer	CLOSED
R21	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (dls_user_data) = "CS_ReqPDU" && status CS Res. = OPEN_CSS start CS Rsp.	OPEN
R22	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu () = "False" && FAL_Pdu_Type (dls_user_data) = "UCS_ReqPDU" UCS_ind { arep_id := GetArepld (), user_data := fal_pdu }	OPEN
R23	OPEN	UCSC Req. finished	OPEN

#	Current State	Event Action	Next State
R24	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu () = "True" (no actions taken)	OPEN
R25	OPEN	FAL-PDU_ind && ((FAL_Pdu_Type (dls_user_data) <> "AbortPDU") (FAL_Pdu_Type (dls_user_data) <> "CS_RspPDU") (FAL_Pdu_Type (dls_user_data) <> "CS_ReqPDU") (FAL_Pdu_Type (dls_user_data) <> "UCS_ReqPDU")) FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "Invalid FAL-PDU", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "Invalid FAL-PDU" additional_detail:= "null" } reset all machines, StopTimer	CLOSED
R26	OPEN	MCTimer expired FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "MCTimer expired", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "MCTimer expired" additional_detail:= "null" }, reset all machines, StopTimer	CLOSED
R27	OPEN	MSTimer expired FAL-PDU_req { dmpm_service_name:= "DMPM_Disconnect_req", arep_id:= GetArepld(), reason:= "MSTimer expired", dlsdu:= "null" }, Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "FAL", reason_code:= "MSTimer expired" additional_detail:= "null" }, reset all machines, StopTimer	CLOSED

#	Current State	Event Action	Next State
R28	OPEN	CS Res. finished <pre> Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Execution error in cyclic data transfer" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Execution error in cyclic data transfer", dlsdu := "null" }, reset all machines, StopTimer </pre>	CLOSED
R29	OPEN	CS Rsp. finished <pre> Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Execution error in cyclic data transfer" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Execution error in cyclic data transfer" dlsdu := "null" }, reset all machines, StopTimer </pre>	CLOSED
R30	NOT CLOSED	<pre> FAL-PDU_ind && dmpm_service_name <> "DMPM_Disconnect_ind" && fal_pdu <> "null" && FAL_Pdu_Type (dls_user_data) = "Abort_PDU" Abort_ind { arep_id:= GetArepld (), locally_generated:= "False", identifier:= AbortIdentifier (fal_pdu), reason_code:= AbortReason (fal_pdu), additional_detail:= AbortDetail (fal_pdu) } reset all machines, StopTimer </pre>	CLOSED
R31	NOT CLOSED	<pre> FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" Abort_ind { arep_id:= GetArepld (), locally_generated:= "False", identifier:= "Data Link Layer", reason_code:= reason additional_detail:= "null" } reset all machines, StopTimer </pre>	CLOSED

#	Current State	Event Action	Next State
R32	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_user" Abort_ind { arep_id:= GetArepld (), locally_generated:= "False", identifier:= "FAL", reason_code:= reason additional_detail:= "null" } reset all machines, StopTimer	CLOSED
R33	NOT CLOSED	FAL-PDU_ind && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" Abort_ind { arep_id:= GetArepld (), locally_generated:= "True", identifier:= "Data Link Layer", reason_code:= reason additional_detail:= "null" } reset all machines, StopTimer CLOSED □ R34 □ NOT CLOSED □ ErrorToARPM (no action taken) NOTE It is a local matter to report this error status to network management entities. The ARPM does not abort the existing connections. The FAL user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive.	SAME

4.6.7.4.3 BNB ARPM State Table (Confirmed Service Sending and Receiving - Client)

Table 126 – BNB ARPM State Table - Sender Transactions (Confirmed Service Sending and Receiving - Client)

#	Current State	Event Action	Next State
S1	OPEN_CSC	CS Req. started && Flag (Buffer_sent_Flag) = "True" SetFlag (Buffer_sent_Flag,False), FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU(fal_pdu_name:= "CS_ReqPDU", fal_data:= user_data) }, StoreIndex (index), SetFlag (Change_Flag,True), StartTimer(MC)	WAIT_SENT_CSC
S2	OPEN_CSC	CS Req. started && Flag (Buffer_sent_Flag) = "False" StoreService(CS_req)	WAIT_RESOURCE

#	Current State	Event Action	Next State
S3	WAIT_RESOURCE_CSC	status UCS Req. <> WAIT_SENT_UCS && status CS Rsp. <> WAIT_SENT_CSS SetFlag (Buffer_sent_Flag,False), RestoreService(CS_req), FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU(fal_pdu_name:= "CS_ReqPDU", fal_data:= user_data) }, StoreIndex (index), SetFlag (Change_Flag,True), StartTimer(MC)	WAIT_SENT_CSC
S4	WAIT_CYC_RES	CS_req && VA_ASE_Service = "Read.req" && TestIndex (index) = "True" FAL-PDU_req { dmpm_service_name:= "DMPM_Get_req", arep_id:= GetArepld(), }	WAIT_CON
S5	WAIT_CYC_RES	CS_req && VA_ASE_Service = "Read.req" && TestIndex (index) = "False" && Flag (Buffer_sent_Flag) = "True" SetFlag (Buffer_sent_Flag,False) FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "CS_ReqPDU", fal_data:= user_data) }, StoreIndex (index), SetFlag (Change_Flag,True)	WAIT_SENT_CSC
S6	WAIT_CYC_RES	CS_req && VA_ASE_Service = "Read.req" && TestIndex (index) = "False" && Flag (Buffer_sent_Flag) = "False" StoreService(CS_req)	WAIT_CYC_RESOURCE
S7	WAIT_CYC_RES	CS_req && VA_ASE_Service <> "Read.req" CS Req. finish	OPEN_CSC

#	Current State	Event Action	Next State
S8	WAIT_CYC RESOURC E	status UCS Req. <> WAIT_SENT_UCS && status CS Rsp. <> WAIT_SENT_CSS SetFlag (Buffer_sent_Flag,False), RestoreService(CS_req), FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU(fal_pdu_name:= "CS_ReqPDU", fal_data:= user_data) }, StoreIndex (index), SetFlag (Change_Flag,True)	WAIT_SENT_ CSC

Table 127 – BNB State Table - Receiver Transactions (Confirmed Service Sending and Receiving - Client)

#	Current State	Event Action	Next State
R1	WAIT_SEN T_CSC	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Sent_ind" && Flag (Buffer_sent_Flag) = "False" SetFlag (Buffer_sent_Flag,True)	WAIT_RES
R2	WAIT_SEN T_CSC	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "True" StartTimer(MC)	WAIT_SENT_ CSC
R3	WAIT_SEN T_CSC	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL_Pdu_Type (dls_user_data) <> "CS_RspPDU" (no actions taken)	WAIT_SENT_ CSC
R4	WAIT_RES	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu () = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "True" CS_con { arep_id := GetArepld (), user_data := fal_pdu }, SetFlag (Change_Flag,False), StartTimer(MC)	WAIT_CYC_ RES
R5	WAIT_RES	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu () = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && Flag (Change_Flag) = "True" && IDM_REQ-Index <> index (no actions taken)	WAIT_RES

#	Current State	Event Action	Next State
R6	WAIT_RES	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "False" && Flag (Change_Flag) = "False" CS Req. finish	OPEN_CSC
R7	WAIT_RES WAITING_CS	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu () = "False" && FAL_Pdu_Type (dls_user_data) <> "CS_RspPDU" (no actions taken)	WAIT_RES
R8	WAIT_CON	FAL-PDU_ind && dmpm_service_name = "DMPM_Get_cnf" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "True" CS_con { arep_id := GetArepld (), user_data := fal_pdu }	WAIT_CYC_RES
R9	WAIT_CON	FAL-PDU_ind && dmpm_service_name = "DMPM_Get_cnf" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "False" CS Req. finish	OPEN_CSC
R10	WAIT_CON	FAL-PDU_ind && dmpm_service_name = "DMPM_Get_cnf" && FAL_Pdu_Type (dls_user_data) <> "CS_ResPDU" (no actions taken)	WAITING_CS
R11	WAITING_CS	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "True" CS_con { arep_id := GetArepld (), user_data := fal_pdu } StartTimer(MC)	WAIT_CYC_RES
R12	WAITING_CS	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" && TestIndex (index) = "False" CS Req. finish	OPEN_CSC
R13	WAIT_CYC_RES WAIT_SENT_CSC WAIT_CON WAIT_CYC_RESOURCE	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL_Pdu_Type (dls_user_data) = "CS_RspPDU" StartTimer(MC)	SAME

4.6.7.4.4 BNB ARPM State Table (Confirmed Service Receiving and Responding - Server)**Table 128 – BNB ARPM State Table - Sender Transactions (Confirmed Service Receiving and Responding - Server)**

#	Current State	Event Action	Next State
S1	WAIT_CYC _IND	CS_rsp && VA_ASE_Service = "Read.rsp" && Flag (Buffer_sent_Flag) = "True" SetFlag (Buffer_sent_Flag,False), FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld (), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "CS_RspPDU", fal_data:= user_data) } }	WAIT_SENT_CSS
S2	WAIT_CYC _IND	CS_rsp && VA_ASE_Service = "Read.rsp" && Flag (Buffer_sent_Flag) = "False" StoreService(CS_req)	WAIT_RESOURCE_CSS
S3	WAIT_CYC _IND	CS_rsp && VA_ASE_Service <> "Read.rsp" CS Rsp. finish	OPEN_CSS
S4	WAIT_RESOURCE_CSS	Closed UCS && ((state CS Req. = OPEN_CSS) (state CS Req. = WAIT_CYC_RES)) SetFlag (Buffer_sent_Flag,False), RestoreService(CS_req) FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU(fal_pdu_name:= "CS_RspPDU", fal_data:= user_data) }, } StartTimer(MS)	WAIT_SENT_CSS

Table 129 – BNB State Table - Receiver Transactions (Confirmed Service Receiving and Responding - Server)

#	Current State	Event Action	Next State
R1	OPEN_CSS	CS Rsp. Started StoreService(FAL-PDU_ind) CS_ind { arep_id:= GetArepld (), user_data:= fal_pdu }, StartTimer(MS)	WAIT_CYC_IND

#	Current State	Event Action	Next State
R2	WAIT_SENT_CSS	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Sent_ind" && Flag (Buffer_sent_Flag) = "False" RestoreService(FAL-PDU_ind), CS_ind { arep_id:= GetArepld (), user_data:= fal_pdu }, SetFlag (Buffer_sent_Flag,True)	WAIT_CYC_IND
R3	WAIT_CYC_IND WAIT_SENT_CSS WAIT_RESOURCE_CSS	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL-PDU_Type (dls_user_data) = "CS_ReqPDU" StoreService(FAL-PDU_ind), CS_ind { arep_id:= GetArepld (), user_data:= fal_pdu }, StartTimer(MS)	SAME
R4	WAIT_CYC_IND WAIT_SENT_CSS WAIT_RESOURCE_CSS	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Received_ind" && dmpm_duplicate_dlsdu = "False" && FAL-PDU_Type (dls_user_data) <> "CS_ReqPDU" (no actions taken)	SAME

4.6.7.4.5 BNB ARPM State Table (Unconfirmed Service Sending - Client)

**Table 130 – BNB ARPM State Table - Sender Transactions
(Unconfirmed Service Sending - Client)**

#	Current State	Event Action	Next State
S1	OPEN_UCSC	UCSC Req. started && Flag (Buffer_sent_Flag) = "True" SetFlag (Buffer_sent_Flag,False), FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "UCS_ReqPDU", fal_data:= user_data) }	WAIT_SENT_UCSC
S2	OPEN_UCSC	UCSC Req. started && Flag (Buffer_sent_Flag) = "False" StoreService(UCS_req)	WAIT_RESOURCE_UCSC
S3	WAIT_RESOURCE_UCSC	state CS Req. = WAIT_SENT_CSC II state CS Rsp. = WAIT_SENT_CSS (no actions taken)	WAIT_RESOURCE_UCSC

#	Current State	Event Action	Next State
S4	WAIT_RESOURCE_UCSC	state CS Req. <> WAIT_SENT_CSC && state CS Rsp. <> WAIT_SENT_CSS SetFlag (Buffer_sent_Flag,False), RestoreService(UCS_req), FAL-PDU_req { dmpm_service_name:= "DMPM_Put_req", arep_id:= GetArepld(), dlsdu:= BuildFAL-PDU (fal_pdu_name:= "UCS_ReqPDU", fal_data:= user_data) } }	WAIT_SENT_UCSC

Table 131 – BNB State Table - Receiver Transactions (Unconfirmed Service Sending - Client)

#	Current State	Event Action	Next State
R1	WAIT_SENT_UCSC	FAL-PDU_ind && dmpm_service_name = "DMPM_Buffer_Sent_ind" && Flag (Buffer_sent_Flag) = "False" SetFlag (Buffer_sent_Flag,True), finish UCSC Req.	OPEN_UCSC

4.6.7.4.6 Functions used by BNB ARPM**Table 132 – Function GetArepld ()**

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 133 – Function dmpm_duplicate_dlsdu ()

Name	dmpm_duplicate_dlsdu ()	Used in	ARPM
Input		Output	
(none)		Boolean	
Function	Returns a value that identifies the duplicated receiving.		

Table 134 – Function BuildFAL-PDU ()

Name	BuildFAL-PDU ()	Used in	ARPM
Input		Output	
fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail		dlsdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 135 – Function FAL_Pdu_Type ()

Name	FAL_Pdu_Type ()	Used in	ARPM
Input		Output	
dls_user_data		One of the FAL-PDU types defined in the clause 4.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FAL-PDU types.		

Table 136 – Function AbortIdentifier ()

Name	AbortIdentifier ()	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 137 – Function AbortReason ()

Name	AbortReason ()	Used in	ARPM
Input		Output	
fal_pdu		The Reason Code parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter.		

Table 138 – Function AbortDetail ()

Name	AbortDetail ()	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function			
This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.			

NOTE The following two functions make use of persistent protocol timers that are able to issue the local events "MSTimer expired" and "MCTimer expired" to notify the expiration of the appropriate time interval to the ARPM.

Table 139 – Function StartTimer ()

Name	StartTimer ()	Used in	ARPM
Input		Output	
identifier			
Function			
This function starts the selected persistent protocol timer as follows: If identifier is MS then function starts the MSTimer with the value of CIN. If identifier is MC then function starts the MCTimer with the value of CIN.			

NOTE The appropriate timer is restarted if it was still running.

Table 140 – Function StopTimer ()

Name	StopTimer ()	Used in	ARPM
Input		Output	
Function	This function stops all local persistent protocol timers of the ARPM.		

NOTE The following two functions make use of local persistent variables to store/restore all parameters of the service request.

Table 141 – Function StopService ()

Name	StoreService ()	Used in	ARPM
Input		Output	
service			
Function	This function stores the service data unit.		

Table 142 – Function RestoreService ()

Name	RestoreService ()	Used in	ARPM
Input		Output	
		service	
Function	This function restores all service parameters. That means the calling ARPM has got access to the service data unit.		

NOTE The following two functions make use of local persistent variables Change_Flag and Buffer_sent_Flag that store the result of checks. The initial value of Change_Flag is False and of Buffer_sent_Flag is True.

Table 143 – Function Flag ()

Name	Flag ()	Used in	ARPM
Input		Output	
identifier		Boolean (True/False)	
Function	This function returns the contents of the identified Flag.		

Table 144 – Function SetFlag ()

Name	SetFlag ()	Used in	ARPM
Input		Output	
identifier			
Boolean (True/False)			
Function	This function sets the identified flag corresponding to the input value.		

NOTE The following two functions make use of local persistent variables that store the passed value.

Table 145 – Function StoreIndex ()

Name	StoreIndex ()	Used in	ARPM
Input		Output	
index			
Function	This function stores the current index.		

Table 146 – Function TestIndex ()

Name	TestIndex ()	Used in	ARPM
Input		Output	
index		Boolean (True/False)	
Function	This function returns the result of the index check.		

4.6.8 Buffered Networkscheduled Unidirectional (BNU) ARPM

4.6.8.1 Primitive Definitions

4.6.8.1.1 Primitives Exchanged between ARPM and FSPM

Table 147 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
UCS_req	FSPM	user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.
FCMP_req	FSPM	(none)	This is an FAL internal primitive used to convey an FAL-Compel (FCMP) request primitive from the FSPM to the ARPM.
GBM_req	FSPM	(none)	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) request primitive from the FSPM to the ARPM.

Table 148 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.
FCMP_cnf	ARPM	arep_id, status	This is an FAL internal primitive used to convey an FAL-Compel (FCMP) confirmation primitive from the ARPM to the FSPM.
GBM_cnf(+)	ARPM	arep_id, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) positive confirmation primitive from the ARPM to the FSPM.
GBM_cnf(-)	ARPM	arep_id	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) negative confirmation primitive from the ARPM to the FSPM.
FSTS_ind	ARPM	arep_id, reported_status	This is an FAL internal primitive used to convey a FAL-Status (FSTS) indication primitive from the ARPM to the FSPM.

4.6.8.1.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 149.

Table 149 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL-SDU parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Link Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.8.2 DLL Mapping of BNU AREP Class

This clause describes the mapping of the BNU AREP class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data Link Layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the BNU AREP class are defined in this clause.

CLASS: Bnu

PARENT CLASS: BufferedNetworkScheduledUnidirectionalAREP

ATTRIBUTES:

- 1 (m) KeyAttribute: PublisherDlcepAddress
- 2 (m) Attribute: DlsapRole (Basic)
- 3 (m) Attribute: QosParameterSet
- 3.1 (m) Attribute: DlcepClass (Publisher, Subscriber)
- 3.2 (m) Attribute: DllPriority (Urgent, Normal, TimeAvailable)
- 3.3 (m) Attribute: DlpduAuthentication (Ordinary, Source)
- 3.4 (m) Attribute: ResidualActivity
- 3.4.1 (m) Attribute: AsSender (False)
- 3.5 (m) Attribute: MaxConfirmDelayOnDIConnect
- 3.6 (m) Attribute: DlschedulingPolicy (Explicit)
- 3.7 (m) Constraint: DlcepClass = Publisher
- 3.7.1 (m) Attribute: FromRequestorToResponder (Ordered, Unordered)
- 3.7.2 (m) Attribute: MaxDlsduSizeFromRequestor
- 3.7.3 (o) Attribute: SenderTimeliness
- 3.7.3.1 (o) Attribute: PublisherDITimelinessClass (Residence, Update, Synchronized, None)
- 3.7.3.2 (o) Attribute: PublisherTimeWindowSize
- 3.7.3.3 (o) Constraint: PublisherDITimelinessClass == Update || Synchronized
- 3.7.3.3.1 (o) Attribute: PublisherSynchronizingDlcep
- 3.8 (m) Constraint: DlcepClass = Subscriber
- 3.8.1 (m) Attribute: FromResponderToRequestor (Ordered, Unordered)
- 3.8.2 (m) Attribute: MaxDlsduSizeFromResponder
- 3.8.3 (o) Attribute: ReceiverTimeliness
- 3.8.3.1 (o) Attribute: PublisherDITimelinessClass (Residence, Update, Synchronized, None)
- 3.8.3.2 (o) Attribute: PublisherTimeWindowSize
- 3.8.3.3 (o) Attribute: SubscriberDITimelinessClass (Residence, Update, Synchronized, None)
- 3.8.3.4 (o) Attribute: SubscriberTimeWindowSize
- 3.8.3.5 (o) Constraint: SubscriberDITimelinessClass == Update || Synchronized
- 3.8.3.5.1 (o) Attribute: SubscriberSynchronizingDlcep
- 3.9 (m) Attribute: LasScheduled (True, False)

DLL SERVICES:

1	(m)	OpsService:	DL-Put
2	(m)	OpsService:	DL-Get
3	(m)	OpsService:	DL-Connect
4	(m)	OpsService:	DL-Connection-Established
5	(m)	OpsService:	DL-Disconnect
6	(m)	OpsService:	DL-Buffer-Received
7	(m)	OpsService:	DL-Buffer-Sent
8	(c)	Constraint:	LasScheduled == False
8.1	(m)	OpsService:	DL-Compel-Service

4.6.8.2.1 Attributes

4.6.8.2.1.1 PublisherDlcepAddress

This attribute specifies the Publisher's DLCEP address and identifies the DLCEP.

The value of this attribute is used as the "DLCEP-address" parameter of the DLL.

This attribute contains the following three subattributes: Link Address, Node Address, and Selector.

NOTE Since the local Link and Node addresses are set by the Network Management, only the Selector portion of the LocalDisapAddress attribute is a configurable attribute of the FAL.

4.6.8.2.1.2 Role

This attribute specifies the role of this AREP. The value of "PushPublisher" indicates that this AREP is used as a publisher. The value of "PushSubscriber" means that it is used as a subscriber.

4.6.8.2.1.3 DisapRole

This attribute specifies the behavior of the DLSAP that is used by the AREP.

This attribute supplies the value for the "DL(SAP)-role" parameter. The possible value Basic corresponds to BASIC as defined in the Data Link Layer standard.

4.6.8.2.1.4 QosParameterSet

DlcepClass

This attribute specifies the behavior of the DLCEP which is attached to the AREP.

This attribute supplies the value for the "DLCEP class" parameter of the DLL. The possible value of this attribute is either Publisher or Subscriber and corresponds to PUBLISHER and SUBSCRIBER defined by the DLL, respectively.

DllPriority

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same BNU AREP.

DlpduAuthentication

This attribute specifies the lower bound of the length of DL-addressing to be used by the DLL.

This attribute supplies the value for the "DLPDU-authentication" parameter of the DLL. The permitted value Ordinary, and Source correspond to ORDINARY and SOURCE, respectively, as defined in the Fieldbus Data Link Layer standard.

ResidualActivity

This attribute specifies receiver's DLC residual activity. It supplies the value for the "Residual activity as receiver" parameter defined in the DLL. The possible values are "True" and "False."

MaxConfirmDelayOnDlConnect

This attribute specifies the maximum confirmation delay for a confirmation from a DL-Connect service.

This attribute supplies the value for the "Maximum confirmation delay on DL-Connect, DL-Reset and DL-Subscriber-Query" parameter specified in the Data Link Layer standard.

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU when it is instructed to do so by a stimulus from the network.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Explicit corresponds to EXPLICIT defined in the Data Link Layer standard (IEC 61158-3).

FromRequestorToResponder

This attribute is used if the Role attribute has a value of “PushPublisher” and specifies the DLL data delivery feature of the AREP.

It supplies the value for the “DLCEP data delivery features from requestor to responder(s)” parameter of the DLL. The possible values Ordered and Unordered correspond to ORDERED and UNORDERED defined in the Data Link Layer standard.

If the DuplicatePduDetectionSupported attribute is True, this attribute has the value of “Ordered.”

MaxDisduSizeFromRequestor

This attribute is used if the Role attribute has a value of “PushPublisher” and specifies the maximum length of an FAL-PDU that can be sent from this AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

FromResponderToRequestor

This attribute is used if the Role attribute has a value of “PushSubscriber” and specifies the DLL data delivery feature of the AREP.

It supplies the value for the “DLCEP data delivery features from responder(s) to requestor” parameter of the DLL. The possible values Ordered and Unordered correspond to ORDERED and UNORDERED defined in the Data Link Layer standard.

If the DuplicatePduDetectionSupported attribute is True, this attribute has the value of “Ordered.”

MaxDisduSizeFromResponder

This attribute is used if the Role attribute has a value of “PushSubscriber” and specifies the maximum length of an FAL-PDU that can be received by this AREP.

This attribute supplies the value for the “Maximum DLSDU size from responder” parameter of the DLL.

SenderTimeliness**PublisherDITimelinessClass**

This optional attribute provides the timeliness class of a Publisher provided by the DLL. This attribute supplies the value for the “DL-timeliness-class” parameter of the DLL. The permitted values Residence, Update, Synchronized, and None correspond to RESIDENCE, UPDATE, SYNCHRONIZED, and NONE defined by the DLL.

PublisherTimeWindowSize

This optional attribute provides time window of a Publisher provided by the DLL. This attribute supplies the value for the “Time window size” parameter of the DLL.

PublisherSynchronizingDlcep

This optional attribute is present when the PublisherDITimelinessClass attribute has the value of Update or Synchronized and provides a DLCEP that is used to generate synchronizing events by the DLL. The FAL user may derive the AREP from which the synchronizing events are delivered by this attribute value. This attribute supplies the value for the “synchronizing DLCEP” parameter of the DLL.

ReceiverTimeliness**SubscriberDITimelinessClass**

This optional attribute provides the timeliness class of a Subscriber provided by the DLL. This attribute supplies the value for the “DL-timeliness-class” parameter of the DLL. The permitted values Residence, Update, Synchronized, and None correspond to RESIDENCE, UPDATE, SYNCHRONIZED, and NONE defined by the DLL.

SubscriberTimeWindowSize

This optional attribute provides time window of a Subscriber provided by the DLL. This attribute supplies the value for the “Time window size” parameter of the DLL.

SubscriberSynchronizingDlcep

This optional attribute is present when the SubscriberDITimelinessClass attribute has the value of Update or Synchronized and provides a DLCEP that is used to generate synchronizing events by the DLL. The FAL user may derive the AREP from which the synchronizing events are delivered by this attribute value. This attribute supplies the value for the “synchronizing DLCEP” parameter of the DLL.

LasScheduled

This attribute specifies whether this AREP is LAS-scheduled or not. The value of True means that this AREP is LAS-Scheduled. The value of False means that it is not LAS-Scheduled.

NOTE This attribute does not affect the operation of the FAL and the DLL. It is used by the FAL users only.

4.6.8.2.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.8.3 BNU AREP State Machine

4.6.8.3.1 BNU ARPM States

The defined states together with their descriptions of the BNU ARPM are listed in Table 150.

Table 150 – BNU ARPM States

State	Description
CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs.
REQUESTING	The AREP has issued an EST_req and waiting for an EST_cnf primitive.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.

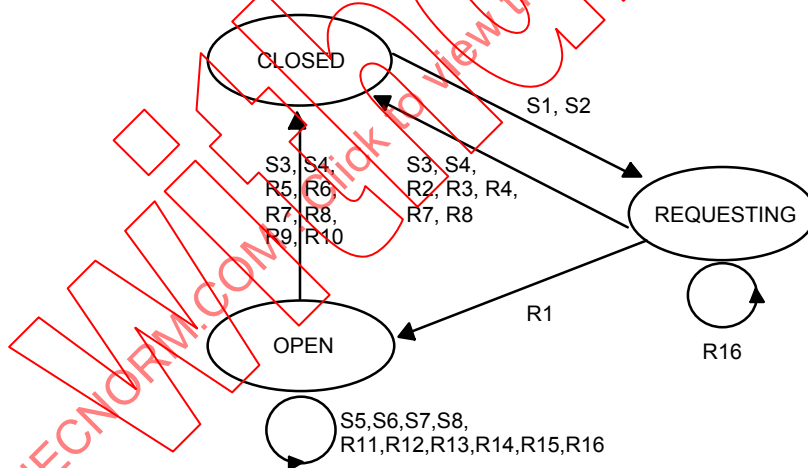


Figure 36 – State Transition Diagram of the BNU ARPM

4.6.8.3.2 BNU ARPM State Table**Table 151 – BNU ARPM State Table - Sender Transactions**

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Role = "PushPublisher" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "null", calling_address := "default dlsap address", local_dlcep_address := PublisherDlcepAddress, dlsdu := "null" }	REQ
S2	CLOSED	EST_req && Role = "PushSubscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := PublisherDlcepAddress, calling_address := "default dlsap address", local_dlcep_address := "default subscriber dlcep address", dlsdu := "null" }	REQ
S3	NOT CLOSED	Abort_req && Role = "PushPublisher" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) }	CLOSED
S4	NOT CLOSED	Abort_req && Role = "PushSubscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := "null" }	CLOSED
S5	OPEN	UCS_req && Role = "PushPublisher" FAL-PDU_req { dmpm_service_name := "DMPM_Put_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_PDU", fal_data := user_data) }	OPEN
S6	OPEN	FCMP_req && Role = "PushPublisher" FAL-PDU_req { dmpm_service_name := "DMPM_Compel_Service_req", arep_id := GetArepld (), action_class := "Local" }	OPEN

#	Current State	Event Action	Next State
S7	OPEN	FCMP_req && Role = "PushSubscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Compel_Service_req", arep_id := GetArepld (), action_class := "Remote" }	OPEN
S8	OPEN	GBM_req && Role = "PushSubscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Get_req", arep_id := GetArepld () }	OPEN

Table 152 – BNU ARPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	REQ	Connect_cnf EST_cnf(+) { arep_id := GetArepld (), user_data := "null" }	OPEN
R2	REQ	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" EST_cnf(-) { arep_id := GetArepld (), user_data := "null" }	CLOSED
R3	REQ	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" EST_cnf(-) { arep_id := GetArepld (), user_data := "null" }	CLOSED
R4	REQ	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name <> "DMPM_Disconnect_ind" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DI Event", additional_detail := "null" }	CLOSED
R5	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" }	CLOSED

#	Current State	Event Action	Next State
R6	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" Abort_ind { arep_id := GetArepld (), locally_generated := "False", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" }	CLOSED
R7	NOT CLOSED	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && Fal_Pdu_Type (fal_pdu) = "Abort_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "False", identifier := AbortIdentifier (fal_pdu), reason_code := AbortReason (fal_pdu), additional_detail := AbortDetail (fal_pdu) }	CLOSED
R8	NOT CLOSED	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && Fal_Pdu_Type (fal_pdu) <> "Abort_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R9	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && (dmpm_service_name <> "DMPM_Buffer_Received_ind") && (dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DMPM_Compel_Service_cnf") && (dmpm_service_name <> "DMPM_Get_cnf") && (dmpm_service_name <> "DMPM_Buffer_Sent_ind") Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DI Event", additional_detail := "null" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }	CLOSED

#	Current State	Event Action	Next State
R10	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_pdu_type <> "UCS_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }	CLOSED
R11	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) = "UCS_PDU" UCS_ind { arep_id := GetArepld (), duplicate_fal_sdu := duplicate_dlsdu, user_data := fal_pdu, local_timeliness := local_dle_timeliness, remote_timeliness := remote_dle_timeliness }	OPEN
R12	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Compel_Service_cnf" FCMP_cnf { status := reason }	OPEN
R13	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Get_cnf" && reason = "success" GBM_cnf(+) { arep_id := GetArepld (), duplicate_fal_sdu := duplicate_dlsdu, user_data := fal_pdu, local_timeliness := local_dle_timeliness, remote_timeliness := remote_dle_timeliness }	OPEN
R14	OPEN	FAL-PDU_ind && Role = "PushSubscriber" && dmpm_service_name = "DMPM_Get_cnf" && reason <> "success" GBM_cnf(-) { arep_id := GetArepld () }	OPEN
R15	OPEN	FAL-PDU_ind && Role = "PushPublisher" && dmpm_service_name = "DMPM_Buffer_Sent_ind" FSTS_ind { arep_id := GetArepld (), reported_status := "Buffer-Sent" }	OPEN

#	Current State	Event Action	Next State
R16	NOT CLOSED	ErrorToARPM (no actions taken) NOTE It is a local matter to report this error status to network management entities. The ARPM does not abort the existing connections. The FAL user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive.	SAME

4.6.8.3.3 Functions used by BNU ARPM

Table 153 – Function GetArepld ()

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 154 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail		dlsdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 155 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
fal_pdu		One of the FAL-PDU types defined in the clause 4.	
Function	This function decodes the FAL-PDU that is conveyed in the fal_pdu parameter and retrieves one of the FAL-PDU types.		

Table 156 – Function AbortIdentifier

Name	AbortIdentifier	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 157 – Function AbortReason

Name	AbortReason	Used in	ARPM
Input		Output	
fal_pdu		The Reason Code parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter.		

Table 158 – Function AbortDetail

Name	AbortDetail	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.		

IECNORM.COM: Click to view the full PDF of IEC 61158-6:2000

4.6.9 Buffered Networkscheduled Unidirectional-MP (BNU-MP) ARPM

4.6.9.1 Primitive Definitions

4.6.9.1.1 Primitives Exchanged between ARPM and FSPM

Table 159 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated Parameters	Functions
EST_req	FSPM	user_data	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
RR . req	FSPM	priority	This is an FAL internal primitive used to convey a Remote Read request primitive from the FSPM to the ARPM
RW . req	FSPM	priority, decoded_buffer_data	This is an FAL internal primitive used to convey a Remote Write request primitive from the FSPM to the ARPM
UCS_req	FSPM	remote_dlsap_address, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.
FCMP_req	FSPM	(none)	This is an FAL internal primitive used to convey an FAL-Compel (FCMP) request primitive from the FSPM to the ARPM.
GBM_req	FSPM	(none)	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) request primitive from the FSPM to the ARPM.

Table 160 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish response(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, remote_disap_address, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.
RR . cnf(+)	ARPM	arep_id, decoded_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey a Remote Read response(+) primitive from the ARPM to the FSPM
RR . cnf(-)	ARPM	arep_id	This is an FAL internal primitive used to convey a Remote Read response(-) primitive from the ARPM to the FSPM
RW . cnf(+)	ARPM	arep_id	This is an FAL internal primitive used to convey a Remote Write response(+) primitive from the ARPM to the FSPM
RW . cnf(-)	ARPM	arep_id	This is an FAL internal primitive used to convey a Remote Write response primitive from the ARPM to the FSPM
FCMP_cnf	ARPM	arep_id, status	This is an FAL internal primitive used to convey an FAL-Compel (FCMP) confirmation primitive from the ARPM to the FSPM.
GBM_cnf(+)	ARPM	arep_id, duplicate_fal_sdu, user_data, local_timeliness, remote_timeliness	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) positive confirmation primitive from the ARPM to the FSPM.
GBM_cnf(-)	ARPM	arep_id	This is an FAL internal primitive used to convey a Get-Buffered-Message (GBM) negative confirmation primitive from the ARPM to the FSPM.
FSTS_ind	ARPM	arep_id, reported_status	This is an FAL internal primitive used to convey an FAL-Status (FSTS) indication primitive from the ARPM to the FSPM.

4.6.9.1.2 Parameters of FSPM/ARPM Primitives

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

Table 161 – Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
duplicate_fal_sdu	This parameter conveys value that is used for the Duplicate_FAL-SDU parameter.
status	This parameter conveys value that is used for the Status parameter.
reported_status	This parameter conveys a Data Link Layer event status.
local_timeliness	This parameter conveys value that is used for the Local_Timeliness parameter.
remote_timeliness	This parameter conveys value that is used for the Remote_Timeliness parameter.

4.6.9.2 DLL Mapping of BNU-MP AREP Class

This clause describes the mapping of the BNU-MP AREP class to the Fieldbus Data Link Layer. It does not redefine the DLSAP attributes or DLME attributes that are or will be defined in the Data Link Layer standard; rather, it defines how they are used by this AR class. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values and the DLL services used with the BNU-MP AREP class are defined in this clause.

CLASS: BnuMp

PARENT CLASS: BufferedNetworkScheduledUnidirectionalAREP

ATTRIBUTES:

1	(m)	KeyAttribute:	PublisherDlcepAddress
2	(m)	Attribute:	DisapRole (Basic)
3	(m)	Attribute:	QosParameterSet
3.1	(m)	Attribute:	DlcepClass (Push Publish, Push Subscriber)
3.2	(m)	Attribute:	DlIPriority (Urgent, Normal, TimeAvailable)
3.3	(m)	Attribute:	DlpduAuthentication (Ordinary, Source)
3.4	(m)	Attribute:	ResidualActivity
3.4.1	(m)	Attribute:	AsSender (False)
3.5	(m)	Attribute:	MaxConfirmDelayOnDIConnect
3.6	(m)	Attribute:	DISchedulingPolicy (Explicit)
3.7	(m)	Constraint:	DlcepClass = Publisher
3.7.1	(m)	Attribute:	FromRequestorToResponder (Ordered, Unordered)
3.7.2	(m)	Attribute:	MaxDisduSizeFromRequestor
3.7.3	(o)	Attribute:	SenderTimeliness
3.7.3.1	(o)	Attribute:	PublisherDITimelinessClass (Residence, Update, Synchronized, None)
3.7.3.2	(o)	Attribute:	PublisherTimeWindowSize
3.7.3.3	(o)	Constraint:	PublisherDITimelinessClass == Update Synchronized
3.7.3.3.1	(o)	Attribute:	PublisherSynchronizingDlcep
3.8	(m)	Constraint:	DlcepClass = Subscriber
3.8.1	(m)	Attribute:	FromResponderToRequestor (Ordered, Unordered)
3.8.2	(m)	Attribute:	MaxDisduSizeFromResponder
3.8.3	(o)	Attribute:	ReceiverTimeliness
3.8.3.1	(o)	Attribute:	PublisherDITimelinessClass (Residence, Update, Synchronized, None)
3.8.3.2	(o)	Attribute:	PublisherTimeWindowSize
3.8.3.3	(o)	Attribute:	SubscriberDITimelinessClass (Residence, Update, Synchronized, None)
3.8.3.4	(o)	Attribute:	SubscriberTimeWindowSize
3.8.3.5	(o)	Constraint:	SubscriberDITimelinessClass == Update Synchronized
3.8.3.5.1	(o)	Attribute:	SubscriberSynchronizingDlcep
3.9	(m)	Attribute:	LasScheduled (True, False)

DLL SERVICES:

1	(m)	OpsService:	DL-Put
2	(m)	OpsService:	DL-Get
3	(m)	OpsService:	DL-Connect
4	(m)	OpsService:	DL-Connection-Established
5	(m)	OpsService:	DL-Disconnect
6	(m)	OpsService:	DL-Buffer-Received
7	(m)	OpsService:	DL-Buffer-Sent
8	(m)	Constraint:	LasScheduled == False
8.1	(m)	OpsService:	DL-Compel-Service

4.6.9.2.1 Attributes

4.6.9.2.1.1 PublisherDlcepAddress

This attribute specifies the Publisher's DLCEP address and identifies the DLCEP. The value of this attribute is used as the "DLCEP-address" parameter of the DLL. This attribute contains a flat addressing.

4.6.9.2.1.2 DisapRole

4.6.9.2.1.3 QosParameterSet

DlcepClass

DlIPriority

NOTE It is not possible to use different priorities for each FAL-PDU sent from the same BNU-MP AREP.

DlPduAuthentication

ResidualActivity

MaxConfirmDelayOnDIConnect

DISchedulingPolicy

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU when it is instructed to do so by a stimulus from the network.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Explicit corresponds to EXPLICIT defined in the Data Link Layer standard (IEC 61158-3).

This attribute provides the guidance to the DLL on the scheduling needed by an AR. For this AREP, the DLL tries to transmit the FAL-PDU when it is instructed to do so by a stimulus from the network.

This attribute supplies the value for the “DL-Scheduling-policy” parameter of the DLL. The permitted value Explicit corresponds to EXPLICIT defined in the Data Link Layer standard (IEC 61158-3).

FromRequestorToResponder

This attribute is used if the Role attribute has a value of “Push Publish” and specifies the DLL data delivery feature of the AREP.

It supplies the value for the “DLCEP data delivery features from requestor to responder(s)” parameter of the DLL. The possible values Ordered and Unordered correspond to ORDERED and UNORDERED defined in the Data Link Layer standard.

If the DuplicatePduDetectionSupported attribute is True, this attribute has the value of “Ordered.”

MaxDisduSizeFromRequestor

This attribute is used if the Role attribute has a value of “Push Publish” and specifies the maximum length of an FAL-PDU that can be sent from this AREP.

This attribute supplies the value for the “Maximum DLSDU sizes from requestor” parameter of the DLL.

FromResponderToRequestor

This attribute is used if the Role attribute has a value of “Push Subscriber” and specifies the DLL data delivery feature of the AREP.

It supplies the value for the “DLCEP data delivery features from responder(s) to requestor” parameter of the DLL. The possible values Ordered and Unordered correspond to ORDERED and UNORDERED defined in the Data Link Layer standard.

If the DuplicatePduDetectionSupported attribute is True, this attribute has the value of “Ordered.”

MaxDisduSizeFromResponder

This attribute is used if the Role attribute has a value of “Push Subscriber” and specifies the maximum length of an FAL-PDU that can be received by this AREP.

This attribute supplies the value for the “Maximum DLSDU size from responder” parameter of the DLL.

SenderTimeliness

PublisherDITimelinessClass

This optional attribute provides the timeliness class of a Publisher provided by the DLL. This attribute supplies the value for the “DL-timeliness-class” parameter of the DLL. The permitted values Residence, Update, Synchronized, and None correspond to RESIDENCE, UPDATE, SYNCHRONIZED, and NONE defined by the DLL.

PublisherTimeWindowSize

This optional attribute provides time window of a Publisher provided by the DLL. This attribute supplies the value for the “Time window size” parameter of the DLL.

PublisherSynchronizingDlcep

This optional attribute is present when the PublisherDITimelinessClass attribute has the value of Update or Synchronized and provides a DLCEP that is used to generate synchronizing events by the

DLL. The FAL user may derive the AREP from which the synchronizing events are delivered by this attribute value. This attribute supplies the value for the “synchronizing DLCEP” parameter of the DLL.

ReceiverTimeliness

SubscriberDITimelinessClass

This optional attribute provides the timeliness class of a Subscriber provided by the DLL. This attribute supplies the value for the “DL-timeliness-class” parameter of the DLL. The permitted values Residence, Update, Synchronized, and None correspond to RESIDENCE, UPDATE, SYNCHRONIZED, and NONE defined by the DLL.

SubscriberTimeWindowSize

This optional attribute provides a time window of a Subscriber provided by the DLL. This attribute supplies the value for the “Time window size” parameter of the DLL.

SubscriberSynchronizingDlcep

This optional attribute is present when the SubscriberDITimelinessClass attribute has the value of Update or Synchronized and provides a DLCEP that is used to generate synchronizing events by the DLL. The FAL user may derive the AREP from which the synchronizing events are delivered by this attribute value. This attribute supplies the value for the “synchronizing DLCEP” parameter of the DLL.

LasScheduled

This attribute specifies whether this AREP is LAS-scheduled or not. The value of True means that this AREP is LAS-Scheduled. The value of False means that it is not LAS-Scheduled.

NOTE This attribute does not affect the operation of the FAL and the DLL. It is used by the FAL users only.

4.6.9.2.2 DLL Services

Refer to 4.7.3, Data Link Layer Service Selection, for DLL service descriptions.

4.6.9.3 BNU-MP ARPM State Machine

4.6.9.3.1 BNU-MP ARPM States

The defined states together with their descriptions of the BNU-MP ARPM are listed in Table 162.

Table 162 – BNU-MP ARPM States

CLOSED	The AREP is defined, but not capable of sending or receiving FAL-PDUs.
REQUESTING	The AREP has issued an EST_req and waiting for an EST_cnf primitive.
ASKING	Transient state before the completion of RR.rq or RW.rq primitive.
GETTING	Transient state before the completion of RR.rq or RW.rq primitive.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.

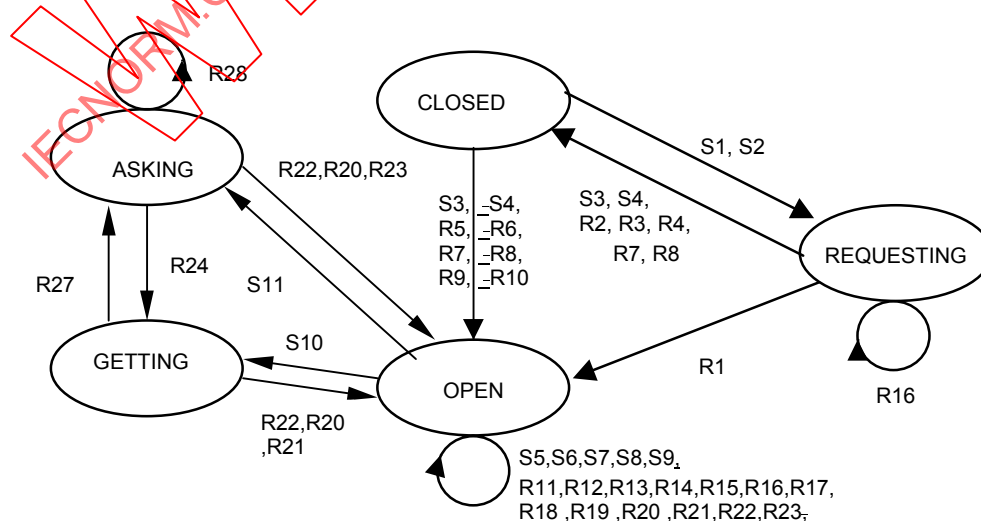


Figure 37 – State Transition Diagram of the BNU-MP ARPM

4.6.9.3.2 BNU-MP ARPM State Table

Table 163 – BNU-MP ARPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Role = "Push Publish" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := "null", calling_address := "default dlsap address", local_dlcep_address := Push DlcepAddress, dlsdu := "null" }	REQ
S2	CLOSED	EST_req && Role = "Push Subscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Connect_req", arep_id := GetArepld (), called_address := PublisherDlcepAddress, calling_address := "default dlsap address", local_dlcep_address := "default subscriber dlcep address", dlsdu := "null" }	REQ
S3	NOT CLOSED	Abort_req && Role = "Push Publish" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := BuildFAL-PDU (fal_pdu_name := "Abort_PDU", fal_id := Identifier, fal_reason_code := reason_code, fal_additional_detail := additional_detail) }	CLOSED
S4	NOT CLOSED	Abort_req && Role = "Push Subscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "disconnection-normal condition", dlsdu := "null" }	CLOSED
S5	OPEN	UCS_req && Role = "Push Publish" && dedicatedArepld() = "False" FAL-PDU_req { dmpm_service_name := "DMPM_Put_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "UCS_PDU", fal_sdu := user_data) }	OPEN
S6	OPEN	FCMP_req && Role = "Push Publish" FAL-PDU_req { dmpm_service_name := "DMPM_Compel_Service_req", arep_id := GetArepld (), action_class := "Local" }	OPEN

#	Current State	Event Action	Next State
S7	OPEN	FCMP_req && Role = "Push Subscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Compel_Service_req", arep_id := GetArepld (), action_class := "Remote" }	OPEN
S8	OPEN	GBM_req && Role = "Push Subscriber" FAL-PDU_req { dmpm_service_name := "DMPM_Get_req", arep_id := GetArepld () }	OPEN
S9	OPEN	UCS_req && Role = "Push Publish" && dedicatedArep() = "True" FAL-PDU_req { dmpm_service_name := "DMPM_Put_req", arep_id := GetArepld (), dlsdu := BuildFAL-PDU (fal_pdu_name := "BufferDataTransfer_PDU", fal_sdu := user_data) }	OPEN
S10	OPEN	RW_req && Role = "Push Publish" && dedicatedArep() = "True" FAL-PDU_req { dmpm_service_name := "DMPM_Put_req", arep_id := GetArepld (), action_class := "local" dlsdu := BuildFAL-PDU (fal_pdu_name := "BufferDataTransfer_PDU", fal_sdu := user_data) }	GETTING
S11	OPEN	RR_req && Role = "Push Subscriber" && dedicatedArep() = "True" action_class := "Remote" FAL-PDU_req { dmpm_service_name := "DMPM_Compel_req", arep_id := GetArepld (), }	ASKING

Table 164 – BNU-MP ARPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	REQ	Connect_cnf <pre> EST_cnf(+) { arep_id := GetArepld (), user_data := "null" } </pre>	OPEN
R2	REQ	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" <pre> EST_cnf(-) { arep_id := GetArepld (), user_data := "null" } </pre>	CLOSED
R3	REQ	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "remote_dls_provider" <pre> EST_cnf(-) { arep_id := GetArepld (), user_data := "null" } </pre>	CLOSED
R4	REQ	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name <> "DMPM_Disconnect_ind" <pre> Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DI Event", additional_detail := "null" } </pre>	CLOSED
R5	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu = "null" && originator = "local_dls_provider" <pre> Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" } </pre>	CLOSED
R6	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && dlsdu = "null" && originator = "remote_dls_provider" <pre> Abort_ind { arep_id := GetArepld (), locally_generated := "False", identifier := "Data Link Layer", reason_code := reason, additional_detail := "null" } </pre>	CLOSED

#	Current State	Event Action	Next State
R7	NOT CLOSED	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && Fal_Pdu_Type (fal_pdu) = "Abort_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "False", identifier := AbortIdentifier (fal_pdu), reason_code := AbortReason (fal_pdu), additional_detail := AbortDetail (fal_pdu) }	CLOSED
R8	NOT CLOSED	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Disconnect_ind" && fal_pdu <> "null" && Fal_Pdu_Type (fal_pdu) <> "Abort_PDU" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }	CLOSED
R9	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && ((dmpm_service_name <> "DMPM_Buffer_Received_ind") && (dmpm_service_name <> "DMPM_Disconnect_ind") && (dmpm_service_name <> "DMPM_Compel_Service_cnf") && (dmpm_service_name <> "DMPM_Get_cnf") && (dmpm_service_name <> "DMPM_Buffer_Sent_ind")) Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid DI Event", additional_detail := "null" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid DL-Event", dlsdu := "null" }	CLOSED
R10	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_pdu_type <> "UCS_PDU" && FAL_pdu_type <> "BufferDataTransfer_PDU", Abort_ind{ arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Invalid FAL-PDU", additional_detail := "null" }, FAL-PDU_req { dmpm_service_name := "DMPM_Disconnect_req", arep_id := GetArepld (), reason := "Invalid FAL-PDU", dlsdu := "null" }	CLOSED

#	Current State	Event Action	Next State
R11	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Buffer_Received_ind" && FAL_Pdu_Type (fal_pdu) = "UCS_PDU BufferDataTransfer_PDU" UCS_ind { arep_id := GetArepld (), duplicate_fal_sdu := duplicate_dlsdu, user_data := fal_pdu, local_timeliness := local_dle_timeliness, remote_timeliness := remote_dle_timeliness }	OPEN
R12	OPEN	FAL-PDU_ind && dmpm_service_name = "DMPM_Compel_Service_cnf" FCPM_cnf { status := dl_status }	OPEN
R13	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Get_cnf" && status = "success" GBM_cnf(+) { arep_id := GetArepld (), duplicate_fal_sdu := duplicate_dlsdu, user_data := fal_pdu, local_timeliness := local_dle_timeliness, remote_timeliness := remote_dle_timeliness }	OPEN
R14	OPEN	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Get_cnf" && status <> "success" GBM_cnf(-) { arep_id := GetArepld () }	OPEN
R15	OPEN	FAL-PDU_ind && Role = "Push Publish" && dmpm_service_name = "DMPM_Buffer_Sent_ind" FSTS_ind { arep_id := GetArepld (), reported_status := "Buffer-Sent" }	OPEN
R18	OPEN	FAL-PDU_ind && Role = "Push Publish" && dmpm_service_name = "DMPM_Put_cnf" && dl_status = "success" FSTS_ind { arep_id := GetArepld (), reported_status := "Put Ok" }	OPEN
R19	OPEN	FAL-PDU_ind && Role = "Push Publish" && dmpm_service_name = "DMPM_Put_cnf" && dl_status <> "success" FSTS_ind { arep_id := GetArepld (), reported_status := dl_status }	OPEN

#	Current State	Event Action	Next State
R20	GETTING, ASKING	FAL-PDU_ind && Role = "Push Subscriber" && (dmpm_service_name = "DMPM_Getcnf" dmpm_service_name = "DMPM_Compel_Service_cnf") && dl_status <> "success" RR cnf(-) { arep_id := GetArepld (), reported_status := dl_status }	OPEN
R21	GETTING G	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Buffer_Received_ind" && dl_status = "success" RR cnf(+) { arep_id := GetArepld (), user_data := fal_pdu, local_timeliness := local_dle_timeliness, remote_timeliness := remote_dle_timeliness }	OPEN
R22	GETTING G, ASKING	FAL-PDU_ind && Role = "Push Publish" && (dmpm_service_name = "DMPM_Putcnf" dmpm_service_name = "DMPM_Compel_Service_cnf") && dl_status <> "success" RW cnf(-) { arep_id := GetArepld (), reported_status := dl_status }	OPEN
R23	ASKING	FAL-PDU_ind && Role = "Push Publish" && dmpm_service_name = "DMPM_Buffer_Sent_ind" && dl_status = "success" RW cnf(+) { arep_id := GetArepld (), reported_status := dl_status }	OPEN
R24	ASKING	FAL-PDU_ind && Role = "Push Subscriber" && dmpm_service_name = "DMPM_Compel_Service_cnf" && dl_status = "success" (no action)	GETTING
R27	GETTING G	FAL-PDU_ind && Role = "Push Publish" && dmpm_service_name = "DMPM_Put_cnf" && dl_status = "success" FAL_PDU_req { dmpm-service-name := "DMPM_compel_service_req" arep_id := GetArepld() action_class := "local" }	ASKING
R28	ASKING	FAL_PDU_Ind && Role = "Push Publish" && dmpm-service-name = "DMPM_Compel_Service_Cnf" && dl_Status = "success" (no action)	Asking

4.6.9.3.3 Functions used by BNU-MP ARPM

Table 165 – Function GetArepld ()

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function	Returns a value that can unambiguously identify the current AREP.		

Table 166 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_pdu_name, calling_dlcep_address, called_dlcep_address, fal_data, fal_id, fal_reason_code, fal_additional_detail		disdu	
Function	Builds an FAL-PDU out of the parameters given as input variables.		

Table 167 – Function FAL_Pdu_Type

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data		One of the FAL-PDU types defined in the clause 4.	
Function	This function decodes the FAL-PDU that is conveyed in the dls_user_data parameter and retrieves one of the FAL-PDU types.		

Table 168 – Function AbortIdentifier

Name	AbortIdentifier	Used in	ARPM
Input		Output	
fal_pdu		The Identifier parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Identifier parameter.		

Table 169 – Function AbortReason

Name	AbortReason	Used in	ARPM
Input		Output	
fal_pdu		The Reason Code parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Reason Code parameter.		

Table 170 – Function AbortDetail

Name	AbortDetail	Used in	ARPM
Input		Output	
fal_pdu		The Additional Detail parameter of the Abort service.	
Function	This function decodes the Abort_PDU that is conveyed in the fal_pdu parameter and extracts the Additional Detail parameter.		

Table 171 – Function dedicatedArep ()

Name	dedicatedArep()	Used in	DMPM
Input		Output	
(none)		True False	
Function	Checks if the AREP is dedicated or not, True means the AREP is dedicated		

4.7 DLL Mapping Protocol Machine (DMPM)

The DLL Mapping Protocol Machine is common to all the AREP types. Only applicable primitives are different among different AREP types.

Remarks about dl identifiers:

1. The IEC Data Link Layer standard defines two types of identifiers to distinguish each DL-primitive or to match one DL outgoing primitive with its mate incoming primitive. They are suffixed as dl-identifier or DLS-user-identifier. In a real implementation of an FAL-DL interface, these identifications may be achieved by means of a pointer to memory location or a return value of a function call, or something else. Since they are purely a local matter, it is not testable over the Fieldbus network. For this reason, they are not included as parameters of the DMPM primitives.
2. “dl-identifiers” and “dls-user-identifiers” are mandatory in the DL-services. The FAL assumes that the values of these parameters are provided from DLSAPs or DLCEPs by a local means.

A remark about DLS-user identification:

It is assumed that a connection between one ARPM instance and one DMPM instance is established locally other than a protocol means. Therefore, DLS-user_identification parameters are not used in the DMPM primitives.

A remark about buffer or queue identifiers:

The IEC Data Link Layer standard uses parameters to identify a queue or a buffer that are shared between the Data Link Layer and the DLS-user. Although they are useful to clarify the operations of the Data Link Layer, none of them affects protocol behavior of the FAL and DL. In a real implementation, these parameters are implementation dependent. Therefore, this standard does not include parameters that directly correspond to these buffer or queue identifiers. A means for identifying the buffers and queues between the FAL and the DL is a local matter.

A remark about initialization of the Data Link Layer:

The IEC Data Link Layer standard defines services to setup resources within the layer, such as DL-Create or DL-Bind services. Although they are useful to clarify the operations of the Data Link Layer, none of them affects protocol behavior of the FAL and DL. Therefore, the FAL assumes that such initialization procedures have been executed prior to the operations of the FAL state machines. For the same reason, these DL-services are not listed in 4.7.3, Data Link Layer Service.

4.7.1 Primitive Definitions

4.7.1.1 Primitives Exchanged between DMPM and ARPM

Table 172 – Primitives issued by ARPM to DMPM

Primitive Names	Source	Associated Parameters	Functions
FAL-PDU_req	ARPM	dmpm_service_name, arep_id, called_address, calling_address, responding_address, local_dlcep_address, reason, action_class, remote_dlsap_address, dll_priority, dls_user_data_timeliness, dlsdu	This primitive is used to request the DMPM to transfer an FAL-PDU, or to request an abort without transferring an FAL-PDU. It passes the FAL-PDU to the DMPM as a DLSDU. It also carries some of the Data Link Layer parameters that are referenced there.

Table 173 – Primitives issued by DMPM to ARPM

Primitive Names	Source	Associated Parameters	Functions
FAL-PDU_ind	DMPM	dmpm_service_name, originator, reason, duplicate_dlsdu, calling_address, calling_dlsap_address, dll_priority, fal_pdu, local_dle_timeliness, remote_dle_timeliness, sender_time_of_production	This primitive is used to pass an FAL-PDU received as a Data Link Layer service data unit to a designated ARPM. It also carries some of the Data Link Layer parameters that are referenced in the ARPM.
Connect_ind	DMPM	calling_dlcep_address, called_dlcep_address, called_address, calling_address, dlcep_class, delivery_from_responder, delivery_from_requestor, dll_priority, max_confirm_delay_on_connect, max_confirm_delay_on_data, dlpdu_authentication, residual_activity_as_sender, residual_activity_as_receiver, dlsdu_size_from_requestor, dlsdu_size_from_responder, sender_dl_timeliness_class, sender_time_of_production, receiver_dl_timeliness_class, dls_user_data	This primitive is used to convey a DL_Connect.ind primitive to the ARPM to process connection establishment. All the parameters that are associated with the DL_Connect.ind primitive are carried with this primitive.
Connect_cnf	DMPM	responding_address, dlcep_class, delivery_from_responder, delivery_from_requestor, dll_priority, max_confirm_delay_on_connect, max_confirm_delay_on_data, dlpdu_authentication, residual_activity_as_sender, residual_activity_as_receiver, dlsdu_size_from_requestor, dlsdu_size_from_responder, sender_dl_timeliness_class, receiver_dl_timeliness_class, sender_time_of_production, dls_user_data	This primitive is used to convey a DL_Connect.cnf primitive to the ARPM. All the parameters that are associated with the DL_Connect.cnf are carried with this primitive.
ErrorToARPM	DMPM	originator, reason	This primitive is used to convey selected communication errors reported by the Data Link Layer to a designated ARPM.

4.7.1.2 Parameters of ARPM/DMPM Primitives

The parameters used with the primitives exchanged between the ARPM and the DMPM are described in table 174.

Table 174 – Parameters used with Primitives Exchanged between ARPM and DMPM

Parameter Name	Description
arep_id	This parameter carries a local identifier to specify the associated AR instance.
action_class	This parameter conveys the value of the dl_action_class parameter.
calling_address	This parameter conveys the value of the dl_calling_address.
calling_dlcep_address	This parameter conveys the value of the RequestingAREP parameter supplied with the EST_ReqPDU.
called_dlcep_address	This parameter conveys the value of the RespondingAREP parameter supplied with the EST_ReqPDU.
called_address	This parameter conveys the value of the dl_called_address parameter.
dmpm_service_name	This parameter conveys a Data Link Layer service name. Possible values are all the DL-XXXX.yyy primitives defined in this section and are represented as DMPM-XXXX.yyy.
dlcep_address	This parameter conveys the value of the dl_dlcep_address parameter.
duplicate_dlsdu	This parameter conveys the value of the dl_duplicate_dlsdu parameter of the received primitive.
dlcep_class	This parameter conveys the value of the dl_dlcep_class parameter.
dll_priority	This parameter conveys the value of the dl_dll_priority parameter.
delivery_from_responder	This parameter conveys the value of the dl_delivery_from_responder parameter.
delivery_from_requestor	This parameter conveys the value of the dl_delivery_from_requestor parameter.
dlsdu_size_from_requestor	This parameter conveys the value of the dl_dlsdu_size_from_requestor parameter.
dlsdu_size_from_responder	This parameter conveys the value of the dl_dlsdu_size_from_responder parameter.
dls_user_data	This parameter conveys the value of the dl_dls_user_data parameter.
dlsdu	This parameter conveys the value of the dl_dls_user_data parameter.
dlpdu_authentication	This parameter conveys the value of the dl_dlpdu_authentication parameter.
fal_pdu	This parameter conveys the value of the dl_dls_user_data parameter.
local_dlsap_address	This parameter conveys the value of the dl_local_dlsap_address parameter.
remote_dle_timeliness	This parameter conveys the value of the dl_sender_and_remote_dle_timeliness parameter.
local_dle_timeliness	This parameter conveys the value of the dl_dle_timeliness parameter.
max_confirm_delay_on_connect	This parameter conveys the value of the dl_max_confirm_delay_on_connect parameter.
max_confirm_delay_on_data	This parameter conveys the value of the dl_max_confirm_delay_on_data parameter.
originator	This parameter conveys the value of the dl_originator parameter.
reason	This parameter conveys the value of the dl_reason parameter.
responding_address	This parameter conveys the value of the dl_responding_address parameter.
residual_activity_as_sender	This parameter conveys the value of the dl_residual_activity_as_sender parameter.
residual_activity_as_receiver	This parameter conveys the value of the dl_residual_activity_as_receiver parameter.
sender_dl_timeliness_class	This parameter conveys the value of the dl_sender_dl_timeliness_class parameter.
receiver_dl_timeliness_class	This parameter conveys the value of the dl_receiver_dl_timeliness_class parameter.
sender_time_of_production	This parameter conveys the value of the dl_time_of_production parameter.
remote_dlsap_address	This parameter conveys the value of the dl_remote_dlsap_address parameter.

4.7.1.3 Primitives Exchanged between Data Link Layer and DMPM

NOTE The following primitives and their parameters are defined in IEC 61158-3.

Table 175 – Primitives Exchanged between Data Link Layer and DMPM

Primitive Names	Source	Associated Parameters and Their Types
DL_Connect.ind	Data Link Layer	dl_called_address, dl_calling_address, dl_dlcep_class, dl_delivery_from_requestor, dl_delivery_from_responder, dl_dll_priority, dl_max_confirm_delay_on_connect, dl_max_confirm_delay_on_data, dl_dlpdu_authentication, dl_residual_activity_as_sender, dl_residual_activity_as_receiver, dl_dlsdu_size_from_requestor, dl_dlsdu_size_from_responder, dl_sender_dl_timeliness_class, dl_sender_time_of_production, dl_receiver_dl_timeliness_class, dl_dls_user_data
DL_Connect.req(out)	Data Link Layer	(none)
DL_Connect.cnf		dl_responding_address, dl_dlcep_class, dl_delivery_from_requestor, dl_delivery_from_responder, dl_dll_priority, dl_max_confirm_delay_on_connect, dl_max_confirm_delay_on_data, dl_dlpdu_authentication, dl_residual_activity_as_sender, dl_residual_activity_as_receiver, dl_dlsdu_size_from_requestor, dl_dlsdu_size_from_responder, dl_sender_dl_timeliness_class, dl_receiver_dl_timeliness_class, dl_sender_dl_timeliness_class, dl_sender_time_of_production, dl_receiver_dl_timeliness_class, dl_dls_user_data
DL_Connection_Established.ind	Data Link Layer	(none)
DL_Disconnect.ind	Data Link Layer	dl_originator, dl_reason, dl_dls_user_data
DL_Data.ind	Data Link Layer	dl_dls_user_data
DL_Data.cnf	Data Link Layer	dl_status
DL_Buffer_Received.ind	Data Link Layer	dl_duplicate_dlsdu
DL_Unitdata.ind	Data Link Layer	dl_called_address, dl_calling_address, dl_dll_priority, dl_dls_user_data
DL_Unitdata.cnf	Data Link Layer	dl_status
DL_Buffer_Sent.ind	Data Link Layer	(none)
DL_Get.req(out)	Data Link Layer	dl_status, dl_reported_service_identification_class, dl_calling_dlsap_address, dl_dll_priority, dl_dls_user_data, dl_local_dle_timeliness, dl_sender_and_remote_dle_timeliness, dl_sender_time_of_production
DL_Put.req(out)	Data Link Layer	dl_status
DL_CompeI_Service.req(out)	Data Link Layer	dl_status

Primitive Names	Source	Associated Parameters and Their Types
DL_Connect.req(in)	DMPM	dl_called_address, dl_calling_address, dl_dlcep_address, dl_dlcep_class, dl_delivery_from_requestor, dl_delivery_from_responder, dl_dll_priority, dl_max_confirm_delay_on_connect, dl_max_confirm_delay_on_data, dl_dlpdu_authentication, dl_residual_activity_as_sender, dl_residual_activity_as_receiver, dl_scheduling_policy, dl_dlsdu_size_from_requestor, dl_dlsdu_size_from_responder, dl_sender_dl_timeliness_class, dl_sender_time_window_size, dl_sender_synchronizing_dlcep, dl_sender_time_of_production, dl_receiver_dl_timeliness_class, dl_receiver_time_window_size, dl_receiver_synchronizing_dlcep, dl_dls_user_data
DL_Connect.rsp	DMPM	dl_responding_address, dl_dlcep_address, dl_dlcep_class, dl_delivery_from_requestor, dl_delivery_from_responder, dl_dll_priority, dl_max_confirm_delay_on_connect, dl_max_confirm_delay_on_data, dl_dlpdu_authentication, dl_residual_activity_as_sender, dl_residual_activity_as_receiver, dl_scheduling_policy, dl_dlsdu_size_from_requestor, dl_dlsdu_size_from_responder, dl_sender_dl_timeliness_class, dl_sender_time_window_size, dl_sender_synchronizing_dlcep, dl_sender_time_of_production, dl_receiver_dl_timeliness_class, dl_receiver_time_window_size, dl_receiver_synchronizing_dlcep, dl_dls_user_data
DL_Unitdata.req	DMPM	dl_called_address, dl_calling_address, dl_dll_priority, dl_max_confirm_delay, dl_remote_dle_confirmed, dl_dls_user_data
DL_Disconnect.req	DMPM	dl_reason dl_dls_user_data
DL_Data.req	DMPM	dl_dls_user_data
DL_Get.req(in)	DMPM	(none)
DL_Put.req(in)	DMPM	dl_dls_user_data, dl_dls_user_data_timeliness
DL_Compel_Service.req(in)	DMPM	dl_action_class, dl_remote_dlsap_address, dl_priority

4.7.1.4 Parameters of DMPM/Data Link Layer Primitives

The parameters used with the primitives exchanged between the DMPM and the Data Link Layer are identical to those defined in the Data Link Layer Service Definition (IEC 61158-3). They are prefixed by “dl_” to indicate that they are used by the FAL.

4.7.2 DMPM State Machine

4.7.2.1 DMPM States

The defined state of the DMPM together with its description are listed in table 176.

Table 176 – DMPM State Descriptions

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

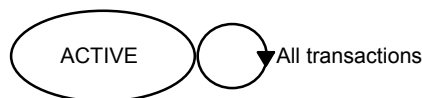


Figure 38 – State Transition Diagram of DMPM

4.7.2.2 DMPM State Table

NOTE1 Although each primitive contains all the available parameters, only those applicable to particular ARPM are relevant.

NOTE2 Parameters starting with a capital letter, "DlcepClass" for instance, refer to those defined in the attribute list of each ARPM. Therefore, they are not conveyed by the service primitives defined here.

Table 177 – DMPM State Table - Sender Transactions

#	Current State	Event Action	Next State
S1	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Connect_req" PickArep (arep_id), DL_Connect.req(in) { dl_called_address := called_address, dl_calling_address := calling_address, dl_dlcep_address := local_dlcep_address, dl_dlcep_class := DlcepClass, dl_delivery_from_requestor := FromRequestorToResponder, dl_delivery_from_responder := FromResponderToRequestor, dl_dl_priority := DlIPriority, dl_max_confirm_delay_on_connect := MaxConfirmDelayOnDIConnect, dl_max_confirm_delay_on_data := MaxConfirmDelayOnDIData, dl_dlpdu_authentication := DlpduAuthentication, dl_residual_activity_as_sender := ResidualActivityAsSender, dl_residual_activity_as_receiver := ResidualActivityAsReceiver, dl_scheduling_policy := DISchedulingPolicy, dl_dlsdu_size_from_requestor := MaxDlsduSizeFromRequestor, dl_dlsdu_size_from_responder := MaxDlsduSizeFromResponder, dl_sender_dl_timeliness_class := PublisherDITimelinessClass, dl_sender_time_window_size := PublisherTimeWindowSize, dl_sender_synchronizing_dlcep := PublisherSynchronizingDlcep, dl_time_of_production := TimeOfProduction, dl_receiver_dl_timeliness_class := SubscriberDITimelinessClass, dl_receiver_time_window_size := SubscriberTimeWindowSize, dl_receiver_synchronizing_dlcep := SubscriberSynchronizingDlcep, dl_dls_user_data := dlsdu }	ACTIVE

#	Current State	Event Action	Next State
S2	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Connect_rsp" PickArep (arep_id), DL_Connect.rsp { dl_responding_address := responding_address, dl_dlcep_address := local_dlcep_address, dl_dlcep_class := DlcepClass, dl_delivery_from_requestor := FromRequestorToResponder, dl_delivery_from_responder := FromResponderToRequestor, dl_dll_priority := DIIPriority, dl_max_confirm_delay_on_connect := MaxConfirmDelayOnDIConnect, dl_max_confirm_delay_on_data := MaxConfirmDelayOnDIData, dl_dlpdu_authentication := DlpduAuthentication, dl_residual_activity_as_sender := ResidualActivityAsSender, dl_residual_activity_as_receiver := ResidualActivityAsReceiver, dl_scheduling_policy := DISchedulingPolicy, dl_dlsdu_size_from_requestor := MaxDlsduSizeFromRequestorNegotiated, dl_dlsdu_size_from_responder := MaxDlsduSizeFromResponderNegotiated, dl_sender_dl_timeliness_class := PublisherDITimelinessClass, dl_sender_time_window_size := PublisherTimeWindowSize, dl_sender_synchronizing_dlcep := PublisherSynchronizingDlcep, dl_sender_time_of_production := TimeOfProduction, dl_receiver_dl_timeliness_class := SubscriberDITimelinessClass, dl_receiver_time_window_size := SubscriberTimeWindowSize, dl_receiver_synchronizing_dlcep := SubscriberSynchronizingDlcep, dl_dls_user_data := dlsdu }	ACTIVE
S3	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Disconnect_req" PickArep (arep_id), DL_Disconnect.req { dl_reason := reason, dl_dls_user_data := dlsdu }	ACTIVE
S4	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Data_req" PickArep (arep_id), DL_Data.req { dl_dls_user_data := dlsdu }	ACTIVE
S5	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Unitdata_req" PickArep (arep_id), DL_Unitdata.req { dl_called_address := RemoteDisapAddress, dl_calling_address := LocalDisapAddress, dl_dll_priority := DIIPriority, dl_max_confirm_delay := MaxConfirmDelayOnUnitdata, dl_remote_dle_confirmed := DleRemoteConf, *1 dl_dls_user_data := dlsdu } *1: Default "False"	ACTIVE

#	Current State	Event Action	Next State
S6	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Put_req" PickArep (arep_id), DL_Put.req(in) { dl_dls_user_data := dlsdu, dl_dls_user_data_timeliness := dls_user_data_timeliness } DL_Put.req(out) && dl_status = "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Put_cnf", reason := dl_status } DL_Put.req(out) && dl_status <> "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Put_cnf", reason := dl_status } ErrorToARPM { originator := "local_dls", reason := dl_status }	ACTIVE
S7	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Get_req" PickArep (arep_id), DL_Get.req(in) { } DL_Get.req(out) && dl_reported_service_identification_class = "NONE" FAL-PDU_ind { dmpm_service_name := "DMPM_Get_cnf", reason := dl_status, calling_dlsap_address := dl_calling_dlsap_address, fal_pdu := dl, local_dle_timeliness := dl_local_dle_timeliness, remote_dle_timeliness := dl_sender_and_remote_dle_timeliness, sender_time_of_production := dl_sender_time_of_production }	ACTIVE
S8	ACTIVE	FAL-PDU_req && dmpm_service_name = "DMPM_Compel_req" PickArep (arep_id), DL_Compel_Service.req(in) { dl_action_class := action_class dl_remote_dlsap_address := remote_dlsap_address, dl_priority := dll_priority } DL_Compel_Service.req(out) FAL-PDU_ind { dmpm_service_name := "DMPM_Compel_Service_cnf", reason := dl_status }	ACTIVE

Table 178 – DMPM State Table - Receiver Transactions

#	Current State	Event Action	Next State
R1	ACTIVE	DL_Connect.ind && LocateAREP (dl_dls_user_data) = "NOT FOUND" DL_Disconnect.req { dl_reason := "AREP Not Found", dl_dls_user_data := "null" }	ACTIVE
R2	ACTIVE	DL_Connect.ind && LocateAREP (dl_dls_user_data) = "NOT Relevant" DL_Disconnect.req { dl_reason := "Invalid AREP Type", dl_dls_user_data := "null" }	ACTIVE
R3	ACTIVE	DL_Connect.ind && LocateAREP (dl_dls_user_data) = "True" Connect_ind { calling_dlcep_address := RequestingAREP,; from dlsdu of DL_Connect.ind called_address := RespondingAREP,; from dlsdu of DL_Connect.ind called_address := dl_called_address, calling_address := dl_calling_address, dlcep_class := dl_dlcep_class, delivery_from_requestor := dl_delivery_from_requestor, delivery_from_responder := dl_delivery_from_responder, dll_priority := dl_dll_priority, max_confirm_delay_on_connect := dl_max_confirm_delay_on_connect, max_confirm_delay_on_data := dl_max_confirm_delay_on_data, dlpdu_authentication := dl_dlpdu_authentication, residual_activity_as_sender := dl_residual_activity_as_sender, residual_activity_as_receiver := dl_residual_activity_as_receiver, dlsdu_size_from_requestor := dl_dlsdu_size_from_requestor, dlsdu_size_from_responder := dl_dlsdu_size_from_responder, sender_dl_timeliness_class := dl_sender_dl_timeliness_class, sender_time_of_production := dl_sender_time_of_production, receiver_dl_timeliness_class := dl_receiver_dl_timeliness_class, dls_user_data := dl_dls_user_data }	ACTIVE
R4	ACTIVE	DL_Connect.cnf && FindAREP () = "False" DL_Disconnect.req { dl_reason := "DLCEP Not Found", dl_dls_user_data := "null" }	ACTIVE
R5	ACTIVE	DL_Connect.cnf && FindAREP () = "True" Connect_cnf { responding_address := dl_responding_address, dlcep_class := dl_dlcep_class, delivery_from_requestor := dl_delivery_from_requestor, delivery_from_responder := dl_delivery_from_responder, dll_priority := dl_dll_priority, max_confirm_delay_on_connect := dl_max_confirm_delay_on_connect, max_confirm_delay_on_data := dl_max_confirm_delay_on_data, dlpdu_authentication := dl_dlpdu_authentication, residual_activity_as_sender := dl_residual_activity_as_sender, residual_activity_as_receiver := dl_residual_activity_as_receiver, dlsdu_size_from_requestor := dl_dlsdu_size_from_requestor, dlsdu_size_from_responder := dl_dlsdu_size_from_responder, sender_dl_timeliness_class := dl_sender_dl_timeliness_class, sender_time_of_production := dl_sender_time_of_production, receiver_dl_timeliness_class := dl_receiver_dl_timeliness_class, dls_user_data := dl_dls_user_data }	ACTIVE
R6	ACTIVE	DL_Connection_Established.ind && FindAREP () = "False" (no actions taken)	ACTIVE

#	Current State	Event Action	Next State
R7	ACTIVE	DL_Connection_Established.ind && FindAREP () = "True" FAL-PDU_ind { dmpm_service_name := "DMPM_Connection_Established_ind", fal_pdu := "null" }	ACTIVE
R8	ACTIVE	DL_Disconnect.ind && FindAREP () = "False" (no actions taken)	ACTIVE
R9	ACTIVE	DL_Disconnect.ind && FindAREP () = "True" FAL-PDU_ind { dmpm_service_name := "DMPM_Disconnect_ind", originator := dl_originator, reason := dl_reason, fal_pdu := dl_dls_user_data }	ACTIVE
R10	ACTIVE	DL_Unitdata.ind && FindAREP (dl_called_address) = "False" (no actions taken)	ACTIVE
R11	ACTIVE	DL_Unitdata.ind && FindAREP (dl_called_address) = "True" && ExplicitQueue = "False" FAL-PDU_ind { dmpm_service_name := "DMPM_Unitdata_ind", calling_address := dl_calling_address, dll_priority := dl_dll_priority, fal_pdu := dl_dls_user_data }	ACTIVE
R12	ACTIVE	DL_Unitdata.ind && FindAREP (dl_called_address) = "True" && ExplicitQueue = "True" DL_Get.req(in) { }	ACTIVE
		DL_Get.req(out) && dl_status = "success" && dl_reported_service_identification_class = "DL(SAP)-ADDRESS" FAL-PDU_ind { dmpm_service_name := "DMPM_Unitdata_ind", calling_address := dl_calling_dlsap_address, dll_priority := dl_dll_priority, fal_pdu := dl_dls_user_data, local_dle_timeliness := dl_local_dle_timeliness, remote_dle_timeliness := dl_sender_and_remote_dle_timeliness, sender_time_of_production := dl_sender_time_of_production }	
		DL_Get.req(out) && dl_status <> "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Unitdata_ind", reason := dl_status }	
		ErrorToARPM { originator := "local_dls", reason := dl_status }	

#	Current State	Event Action	Next State
R13	ACTIVE	DL_Unitdata.cnf && dl_status <> "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Unitdata_cnf", reason := dl_status } ErrorToARPM { originator := "local_dls", reason := dl_status }	ACTIVE
R14	ACTIVE	DL_Unitdata.cnf && dl_status = "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Unitdata_cnf", reason := dl_status }	ACTIVE
R15	ACTIVE	DL_Data.ind && FindAREP () = "False" (no actions taken)	ACTIVE
R16	ACTIVE	DL_Data.ind && FindAREP () = "True" && ExplicitQueue = "False" FAL-PDU_ind { dmpm_service_name := "DMPM_Data_ind", fal_pdu := dl_dls_user_data }	ACTIVE
R17	ACTIVE	DL_Data.ind && FindAREP () = "True" && ExplicitQueue = "True" DL_Get.req(in) { } DL_Get.req(out) && dl_status = "success" && dl_reported_service_identification_class = "DLCEP" FAL-PDU_ind { dmpm_service_name := "DMPM_Data_ind", fal_pdu := dl_dls_user_data } DL_Get.req(out) && dl_status <> "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Get_cnf", reason := dl_status } ErrorToARPM { originator := "local_dls", reason := dl_status }	ACTIVE
R18	ACTIVE	DL_Data.cnf && dl_status = "success" (no actions taken)	ACTIVE

#	Current State	Event Action	Next State
R19	ACTIVE	DL_Data.cnf && dl_status <> "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Data_cnf", reason := dl_status } ErrorToARPM { originator := "local_dls", reason := dl_status }	ACTIVE
R20	ACTIVE	DL_Buffer_Received.ind && FindAREP () = "False" (no actions taken)	ACTIVE
R21	ACTIVE	DL_Buffer_Received.ind && FindAREP () = "True" DL_Get.req(in) { } DL_Get.req(out) && dl_status = "success" && dl_reported_service_identification_class = "NONE" FAL-PDU_ind { dmpm_service_name := "DMPM_Buffer_Received_ind" duplicate_dlsdu := dl_duplicate_dlsdu, ; from DL_Buffer_Received.ind fal_pdu := dl_dls_user_data, local_dle_timeliness := dl_local_dle_timeliness, remote_dle_timeliness := dl_sender_and_remote_dle_timeliness, sender_time_of_production := dl_sender_time_of_production } DL_Get.req(out) && dl_status <> "success" FAL-PDU_ind { dmpm_service_name := "DMPM_Get_cnf", reason := dl_status } ErrorToARPM { originator := "local_dls", reason := dl_status }	ACTIVE
R22	ACTIVE	DL_Buffer_Sent.ind && FindAREP () = "False" (no actions taken)	ACTIVE
R23	ACTIVE	DL_Buffer_Sent.ind && FindAREP () = "True" FAL-PDU_ind { dmpm_service_name := "DMPM_Buffer_Sent_ind" }	ACTIVE

4.7.2.3 Functions used by DMPM

Table 179 – Function PickArep

Name	PickArep	Used in	DMPM
Input		Output	
arep_id		(all the attributes of the specified AREP)	
Function	Selects the attributes for the AREP specified by the arep_id parameter. After this function is executed, the attributes of the selected AREP are available to the state machine.		

Table 180 – Function FindAREP

Name	FindAREP	Used in	DMPM
Input		Output	
dl_called_address (local mapping)		True False	
Function	This function identifies the AREP that shall be bound with an active DMPM. True means the AREP exists. If it does, this function also returns a means to send a DMPM primitive to that AREP.		

Table 181 – Function LocateArep

Name	LocateArep	Used in	DMPM
Input		Output	
dl_dls_user_data (of DL_Connect.ind)		True Not Found Not Relevant Access to attributes of the located AREP.	
Function	This function returns the value of True and a means to get access to the attributes of the located AREP if all of the following conditions are met. Otherwise, it returns the value of either "Not Found" or "Not Relevant." - Decodes the dlsdu that is conveyed by the dl_dls_user_data argument and checks if the FAL PDU type is "EST_ReqPDU." - Decodes the FAL-PDU and extracts the RequestingAREP and RespondingAREP parameters. - Looks for the AREPs that can accept a peer-to-peer connection, such as a QUB AREP, whose LocalDlcepAddress attribute value is equal to the RespondingAREP.		

4.7.3 Data Link Layer Service Selection

4.7.3.1 Introduction

This section briefly describes the Data Link Layer services utilized by the FAL. These Data Link Layer services are fully defined in the Data Link Layer standard (IEC 61158-3).

NOTE The FAL assumes that resources, such as buffers or queues, are set up prior to any operations of FAL protocol machines by a local means. DL services such as Create or Bind may be used for this purpose, the use of these services is not required by the FAL. Therefore, these local services are not listed in this section.

4.7.3.1.1 DL-Connect

This service is used to establish a new connection or join in an existing one, and refers to the Connect service.

4.7.3.1.2 DL-Connection-Established

The DL-Connection-Established service is used to inform the FAL which has sent an ASC.rsp(+) primitive that a Data Link Layer connection is ready to use.

4.7.3.1.3 DL-Disconnect

This service is used to release an existing connection or leave from it and refers to the Disconnect service.

4.7.3.1.4 DL-Unitdata

This service is used for the connectionless data transfer mode and refers to the Unitdata service. It is used to transmit an FAL-PDU from one AREP.

4.7.3.1.5 DL-Data

This service is used for connection-oriented data transfer mode and refers to the Data service. This service is used to transfer an FAL-PDU from one AREP.

4.7.3.1.6 DL-Put

This service is used to copy an FAL-PDU to a buffer. It refers to the Put service.

4.7.3.1.7 DL-Get

This service is used to read an FAL-PDU from a buffer, or to attempt to remove an FAL-PDU from a FIFO queue, which has been bound to a DLSAP. It refers to the Get service.

4.7.3.1.8 DL-Buffer-Received

The DL-Buffer-Received service is used to inform the FAL of a new update on the specified receive buffer.

4.7.3.1.9 DL-Buffer-Sent

The DL-Buffer-Sent service is used to inform the FAL that the specified buffer content has just been sent.

4.7.3.1.10 DL-Compel-Data

The DL-Compel-Data service is used to compel transmission of the content of the buffer of the specified DLCEP.

4.8 Protocol Options

4.8.1 Protocol Option 1

4.8.1.1 Introduction

This section provides a guideline for migration from a previous national standard to the IEC FAL. Although some protocol machines as well as encoding rules are selected by this option, this standard does not intend to endorse such selection or restrict other possible selections.

4.8.1.2 Abstract Syntax Selection

This profile uses the following abstract syntax:

- FAL-AR PDU Abstract Syntax 2

4.8.1.3 Protocol Machine Overview

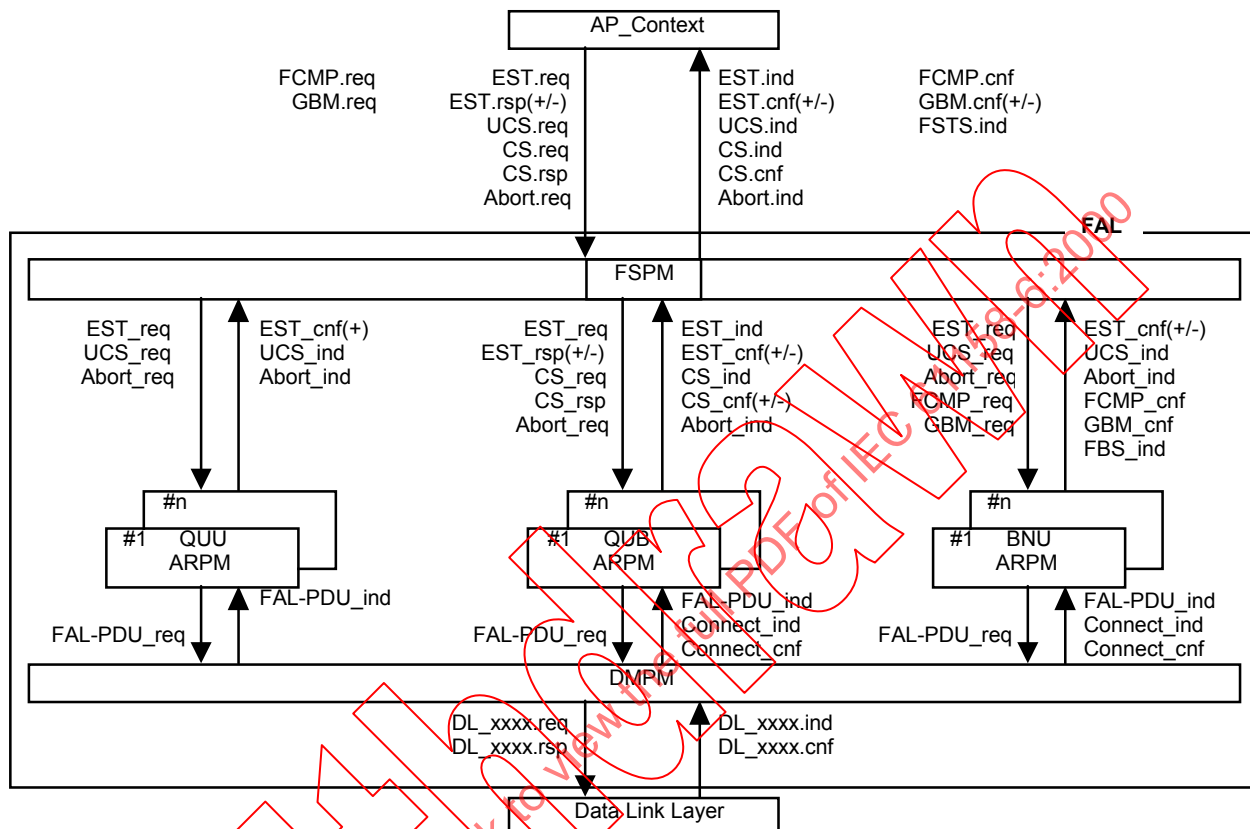


Figure 39 – Primitives Exchanged between Protocol Machines

4.8.1.4 Application Relationship Protocol Machine (ARPM) Selection

The following ARPMs are selected for this profile:

- Queued Usertriggered Unidirectional ARPM (QUU)
- Queued Usertriggered Bidirectional ARPM (QUB)
- Buffered Network-scheduled Unidirectional (BNU)

4.8.1.5 Encoding Rule Selection

This profile uses the following encoding rule:

Traditional Encoding Rule (TER)

4.8.2 Protocol Option 2

4.8.2.1 Introduction

This subclause provides a guideline for migration from a previous national standard to the IEC FAL. Although some protocol machines as well as encoding rules are selected by this option, this standard does not intend to endorse such selection or restrict other possible selections.

4.8.2.2 Abstract Syntax

This profile uses the following abstract syntax:

- FAL-AR PDU Abstract Syntax 1

4.8.2.3 Protocol Machine Overview

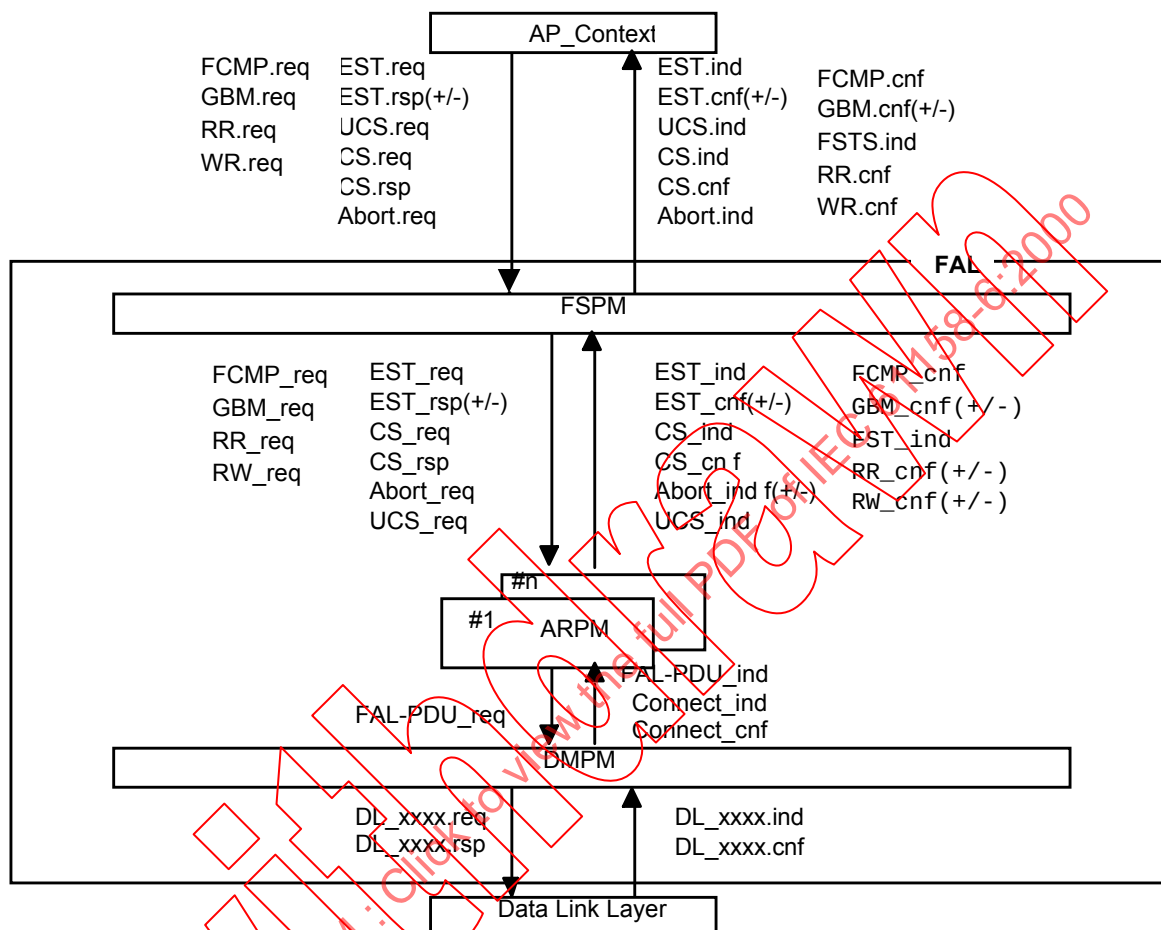


Figure 40 – Primitives Exchanged between Protocol Machines

4.8.2.3.1 Application Relationship Protocol Machine (ARPM) Selections

The following ARPMs are used for this profile:

- Queued Usertriggered Bidirectional-Connectionless (QUB-CL)
- Queued Usertriggered Bidirectional-Segmentation (QUB-SEG)
- Buffered Network-scheduled Unidirectional-MP (BNU-MP)
- Buffered Usertriggered Bidirectional (BUB)
- The other ARPMs defined in this standard can be used.

4.8.2.3.2 Encoding Rule Selection

This profile uses the following encoding rules:

Buffer Encoding Rule (BER) and Messaging Encoding Rule (MER)

4.8.3 Protocol Option 3

4.8.3.1 Introduction

This section provides a guideline for migration from a previous national standard to the IEC FAL. Although some protocol machines as well as encoding rules are selected by this option, this standard does not intend to endorse such selection or restrict other possible selections.

4.8.3.2 Abstract Syntax

This profile uses the following abstract syntax:

- FAL-AR PDU Abstract Syntax 2

4.8.3.3 Protocol Machine Overview

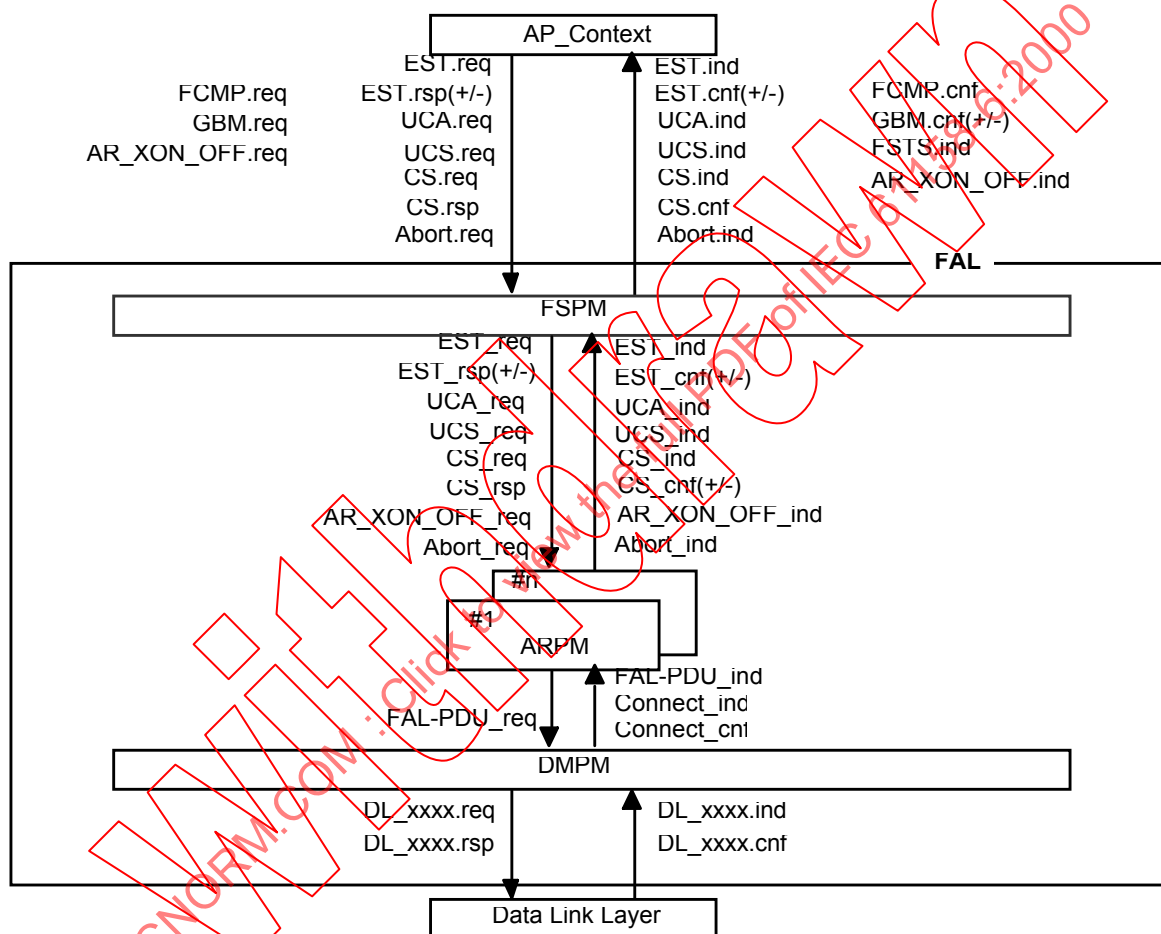


Figure 41 – Primitives Exchanged between Protocol Machines

4.8.3.3.1 Application Relationship Protocol Machine (ARPM) Selection

The following ARPMs are selected for this profile:

- Queued Usertriggered Bidirectional-Flow Control (QUB-FC)
- Buffered Network-scheduled Bidirectional (BNB)

4.8.3.3.2 Encoding Rule Selection

This profile uses the following encoding rule:

- Traditional Encoding Rule (TER)

5 Type 2

5.1 Type 2 Abstract Syntax

5.1.1 FAL PDU Abstract Syntax

FAL_PDU could be either a UCMM_PDU or a Transport_PDU.

UCMM_PDU is used to convey information for connection-less services.

Transport_PDU is used to convey information for connection-oriented services.

A connected message assumes previously negotiated resources and parameters at its source, its destination(s), and any intermediate transit points. These resources are referenced by a unique connection identifier and do not need to be contained in each message, only the connection ID is needed to identify the message and refer to its related parameters, thus giving significant savings in message efficiency.

An unconnected message provides a means to communicate on the local link without previously negotiated resources at the destination so it shall carry full destination ID details, internal data descriptors and full source ID details if a reply is requested. Unconnected messages are used mainly to create connections.

The unconnected service is provided by the Unconnected Message Manager (UCMM). Messages received through the UCMM are forwarded to the Message Router (MR), which direct them to the appropriate internal object for execution. Connections are established by specific unconnected messages sent through the Connection Manager (CM) using UCMM services. Connections may be established either to the Message Router (for messaging purpose), or directly to an application object. Connection target is specified using a connection path, and may be either on the local or a remote link, through several intermediate router nodes. Once a connection is established with an application object, the UCMM, MR and CM are no longer required, since data will be exchanged directly with the connected object, based on the corresponding connection ID. Connected messages sent to the Message Router will be forwarded to the appropriate internal object for execution.

5.1.1.1 PDU Structure

5.1.1.1.1 UCMM_PDU Structure

UCMM_PDU is a sequence of a UCMM_Header, followed by either OM_Service (Object Management ASE) or CM_Service (Connection Manager ASE).

OM_Service could be either OM_Request or OM_Response.

OM_Request consists of MR_Request_Header and a Service_RequestPDU.

Service_RequestPDU could be one of the following :

- Get_Attribute_Single_RequestPDU
- Get_Attribute_All_RequestPDU
- Get_Attribute_List_RequestPDU
- Set_Attribute_Single_RequestPDU
- Set_Attribute_All_RequestPDU
- Set_Attribute_List_RequestPDU
- Reset_RequestPDU
- Create_RequestPDU
- Delete_RequestPDU
- Start_RequestPDU
- Stop_RequestPDU
- Find_Next_Object_Instance_RequestPDU

OM_Response consists of MR_Response_Header and a Service_ResponsePDU.

Service_ResponsePDU could be one of the following :

- Get_Attribute_Single_ResponsePDU
- Get_Attribute_All_ResponsePDU
- Get_Attribute_List_ResponsePDU
- Set_Attribute_Single_ResponsePDU
- Set_Attribute_All_ResponsePDU
- Set_Attribute_List_ResponsePDU
- Reset_ResponsePDU
- Create_ResponsePDU
- Delete_ResponsePDU
- Start_ResponsePDU
- Stop_ResponsePDU
- Find_Next_Object_Instance_ResponsePDU

CM_Service could be either CM_Request or CM_Response.

CM_Request consists of MR_Request_Header and a FWD_RequestPDU.

FWD_Request could be either FWD_Open_RequestPDU or FWD_Close_RequestPDU.

CM_Response consists of MR_Response_Header and a FWD_ResponsePDU.

FWD_Response could be either FWD_Open_ResponsePDU or FWD_Close_ResponsePDU.

5.1.1.1.2 Transport_PDU Structure

Transport_PDU is a sequence of a Transport_Header, followed by either OM_Service (Object Management ASE) or Application Data.

OM_Service is defined in 5.1.1.1 above.

Application Data comes from the source to which the connection has been made.

5.1.1.2 UCMM_PDUs

All UCMM_PDUs shall be sent and received using fixed tag 0x83 or Management tag 0x88. When a device becomes powered, the UCMM shall invoke the DLL_enable_fixed_request service of the Data Link Layer to request that fixed tag 0x83 or 0x88 Lpackets be routed to the UCMM. All sending and receiving of T-PDUs shall use the DLL_fixed_request.req and DLL_fixed_request.ind service primitives of the Data Link Layer, respectively. The packet parameter of these services shall be prefixed with the following header to form the Transport PDU before sending to the Data Link Layer. The format of the UCMM_PDU Header is shown in Table 182.

Table 182: UCMM_PDU Header Format

Parameter Name	Format
command_code	USINT
Timeout	USINT
TransactionID	UINT

The command_code shall specify the type of UCMM_PDU as shown in Table 183. UCMM packets received with a reserved command_code shall be discarded without acknowledgement.

Table 183: UCMM command codes

Command_code	Description
0	Reserved
1	acknowledge a request
2	request with retry until acknowledged
3	response with retry until acknowledged
4	request with no acknowledge and no response
5	acknowledge a response
6	response which shall not retry (no acknowledge)
7	request with retry until response (no acknowledge)
8	Request which shall not retry and shall cause a code 6 response
9 - 255	Reserved

The timeout field shall specify the duration of the transaction in an 8-bit floating-point format. The most significant 5 bits of the field shall be an unsigned exponent that is not biased. The least significant 3 bits shall be the least significant 3 bits of a 4 bit unsigned mantissa. The most significant bit of the mantissa shall be 1 and shall not appear explicitly in the 8-bit representation. The binary point shall be positioned between the implied 1 and the rest of the mantissa. The units of the transaction duration computed using the timeout field shall be in milliseconds.

NOTE Using ANSI C precedence rules, the number of 0,125ms ticks is $((8 | \text{timeout} \& 7) \ll (\text{timeout} \gg 3 \& 31))$.

The transactionID field shall be composed of two sub-fields:

- the least significant 10 bits, RECORD, shall identify a specific transaction;
- the most significant 6 bits, SEQUENCE, shall be used for duplicate detection. It shall be incremented for every unique message on this RECORD; however it shall not be incremented on a retry.

Only one message shall be outstanding for a given RECORD.

If multiple outstanding messages are required then multiple RECORDs are required.

When an I'm alive fixed tag packet is received from a node (see IEC 61158-4, Station Management), all UCMM transactions with that node shall be aborted.

5.1.1.3 Transport Headers

Contents of Transport headers varies depending on the class of transport selected during the connection establishment.

5.1.1.3.1 Class 0 (Null or Base)

The class 0 transport header shall be an unsigned 16-bit number. This header shall be written to the T-PDU buffer by the transport and shall be simply discarded by the transport from the packet coming from the consumer. The format of the Header is shown in Table 184.

Table 184: Transport Class 0 Header

Parameter Name	Format
dont_care	UINT

5.1.1.3.2 Class 1 (Duplicate Detection)

The class 1 transport header shall be an unsigned 16-bit sequence count. This header shall be written to the T-PDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 185.

Table 185: Transport Class 1 Header

Parameter Name	Format
sequence_count	UINT

5.1.1.3.3 Class 2 (Acknowledged)

Like the class 1 transport header, the class 2 transport header shall be a 16-bit sequence count. This header shall be written to the T-PDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 186.

Table 186: Transport Class 2 Header

Parameter Name	Format
sequence_count	UINT

5.1.1.3.4 Class 3 (Verified)

Like the class 1 transport header, the class 3 transport header shall be a 16-bit sequence count. This header shall be written to the T-PDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 187.

Table 187: Transport Class 3 Header

Parameter Name	Format
sequence_count	UINT

5.1.1.3.5 Class 4 (Non-Blocking), Class 5 (Non-Blocking, Fragmenting) and Class 6 (Multicasting, Fragmenting)

These classes of transports all use the header format shown in Table 188. Classes 4 and 5 have two headers in each T-PDU, one Sender and one Receiver. Class 6 uses a Sender header in the Client to Server direction and a Receiver header in the Server to Client direction.

Table 188: Classes 4 to 6 header format

Command			Sequence number	Data	
MSB	Bits 15 – 12		Bits 11 - 8	Bits 7 - 0	LSB
Value :	Meaning in Sender Header:	Meaning in Receiver Header:	Increments from 0 to 15, then rolls over to zero again. Used in Sender header to distinguish a retry from a new transmission. Used in Receiver header to indicate which sender T-PDU is being ACK'd or NAK'd.	For "First": Number of 256 byte blocks (minus 1) to allocate to hold complete message. For "NAK": Reason code. For others: Set to zero when generating; ignore when receiving.	
0.	Null	ACK			
1.	Only	NAK			
2.	First	Not Initialised			
3.	Middle	Undefined			
4.	Last	Undefined			
5. to 15 are undefined					

The Command identifies the use of the T-PDU by the transport. The Command field is defined differently for the Sender and the Receiver header.

Sender header commands :

Null = no data in this Transport PDU

Only = this is the Transport PDU conveying all of the Application data

First / Middle / Last = position of this Transport PDU within the segmented Application data

The Sequence Number is used in the Sender header to distinguish a retry from a new transmission. It is used in Receiver header to indicate which sender T-PDU is being ACK'd or NAK'd. This is **NOT** used for applications request/response matching. This is only used by the transport state machines.

The Data is used by one Sender command and one Receiver command. It is used in the *First* Sender command to indicate the number of 256 byte blocks (minus 1) the Receiver shall allocate for the complete application message. That is, application messages of size 0 to 256 bytes shall have the *First Data* field set to 0; 257 to 512, to 1; and so on up to 65536 bytes. It is used in the NAK Receiver command to provide a reason code:

Class 4 (the Point-to-Point non-Blocking transport) uses two headers in each T-PDU, followed by the application data. The Sender Header is first in the T-PDU, the Receiver Header is second, followed by the application data. Class 4 only uses the *Null* and *Only* Sender Commands (numbered 0 and 1).

Class 5 (the Point-to-Point Fragmenting non-Blocking transport) uses two headers in each T-PDU, followed by the application data. The Sender Header is first in the T-PDU, the Receiver Header is second, followed by the application data. Class 5 uses all of the Sender Commands.

Class 6 (the Multicasting Fragmenting transport) uses one header in each T-PDU, followed by the application data. Class 6 uses all of the Sender Commands. The Sender Header is used in the Client-to-Server T-PDU and the Receiver Header is used in the Server to Client T-PDU.

5.1.1.3.6 Exclusive Owner O⇒T Data Format

Exclusive owner connections shall have either of two real-time transfer formats:

- 32-bit header, fixed size;
- no header, variable size.

The 32-bit header prefixed to the real-time data shall be in the form shown in Table 189.

Table 189: Real-time Data Header

Parameter	Format	Size (bits)
Run_idle	RUN_IDLE	1
Reserved	UDINT	31

The *run_idle* flag (bit 0) shall be set (1 = RUN) to indicate that the following data shall be sent to the target application. It shall be clear (0 = IDLE) to indicate that the idle event shall be sent to the target application. The reserved field (bits 1-31) shall be reserved and set to 0.

If the *no_header* transfer format is used, the reception of a packet with data beyond the transport header shall indicate the RUN mode. If the packet is truncated after the transport header, the target shall be sent an idle event.

5.1.1.4 CM_PDUs

Forward services are provided by the Connection Manager. They are :

CM_FWD_Open

CM_FWD_Close

5.1.1.4.1 Connection Manager

The Connection Manager Object is used to manage the establishment and maintenance of communication connections.

The Connection Manager objects at different nodes shall communicate using the UCMM services described in IEC 61158-5. The T-PDUs with which peer Connection Managers communicate shall be derived from the services of the Connection Manager.

5.1.1.4.2 Forward_Open

The response to the Forward_Open_Request_PDU is a Forward_Open_ResponsePDU. The Forward_Open response shall contain the original connection serial number and owner vendor and serial number and the connection IDs (CID) necessary to complete the connection if successful and error information if the connection failed.

NOTE The Forward_Open request sets up network, transport, and application connections. An application connection consists of a single transport connection, and one or two network connections that are in turn comprised of multiple link connections. Each port segment in the connection path uses a link connection. The Forward_Open service between two devices builds one or two link connections as specified by the network connection parameter and the requested packet intervals (CM_RPI). Since up to two network connections can be required for a single transport connection, they are differentiated by the O⇒T and T⇒O designations; O⇒T means originator to target, and T⇒O means target to originator.

5.1.1.4.2.1 Format of Forward_Open Request

The format of the Forward_Open request T-PDU shall be of the form shown in Table 190:

Table 190: Forward_Open Request Format

Parameter Name	Format
priority_and_tick	USINT
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_connection_parameters	UINT
T2O_CM_RPI	UDINT
T2O_connection_parameters	UINT
xport_type_and_trigger	USINT
connection_path_size	USINT
connection_path[]	UINT

5.1.1.4.2.2 Format of Forward_Open Response if Success

If the status parameter of the CM_open_response is zero (no error), the format of the Forward_Open response T-PDU shall be of the form shown in Table 191.

Table 191: Forward_Open_Good Response Format

Parameter Name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (in 16-bit words)	USINT
Reserved	USINT
application_reply[]	UINT

Success shall be returned when the connection requested has been established from this point forward in the path. This response also shall indicate the connection serial number and the actual packet rate of the connection. Once the successful response has been received, the connection shall be open from this point forward in the path. Targets shall wait at least 10 seconds after sending the CM_Forward_Open_Good_Response for the first packet on a connection.

5.1.1.4.2.3 Format of Forward_Open Response if Failure

If the status parameter of the CM_open_response is not zero (error), the format of the Forward_Open response T-PDU shall be of the form shown in Table 192.

Table 192: Forward_Open_Bad Response Format

Parameter Name	Format
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (in 16-bit words)	USINT
Reserved	USINT

This format shall be used for all failures of the connection system. The requested connection shall not be established, and the object specific status words shall contain information about the reason for the failure. The remaining_path_size shall contain the length of the path at the point the connection failed. This information can be used to debug the problem.

In the failure response, the remaining remaining_path_size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining remaining_path_size.

A duplicate Forward_Open service shall be defined as a Forward_Open service whose vendor_ID, connection_serial_number, and originator_serial_number match an existing connection's parameters. If the duplicate Forward_Open service is a null Forward_Open service (defined as the connection type in both the O_T and T_O network connection parameter fields are NULL), then the

Forward_Open service shall be forwarded to the application for further processing. Null Forward_Open requests may be used to reconfigure the connection. The Connection Manager in the intermediate nodes need not allocate additional resources for a duplicate Forward_Open request since the resources have already been allocated. If the duplicate Forward_Open request is not NULL, then an general status = 0x01, extended status = 0x0100 shall be returned.

5.1.1.4.3 Forward_Close

The Forward_Close request shall remove a connection from all the nodes participating in the original connection. The Forward_Close shall be sent between Connection Managers as specified in the connection_path. The Forward_Close request shall cause all resources in all nodes participating in the connection to be deallocated, including connection IDs, link transmit time, and internal memory buffers.

If an intermediate node cannot find the connection that is to be closed (it may have timed out at the node), the Forward_Close request shall still be forwarded to downstream nodes or the target application (via the CM_close_indication).

NOTE The Forward_Close is always forwarded to allow downstream nodes or the target application to release any resources that were allocated for the connection..

5.1.1.4.3.1 Format of Forward_Close Request

The format of the Forward_Close request shall be of the form shown in Table 193.

Table 193: Forward_Close Request Format

Parameter Name	Format
connection_priority/tick time	USINT
connection_timeout	USINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection_path_size	USINT
Reserved	USINT
connection_path[]	UINT

5.1.1.4.3.2 Format of Forward_Close Response if Success

The Forward_Close response shall be sent between Connection Managers in response to a Forward_Close request, and shall contain the status of the close request. If the status parameter of the CM_close_response is zero (no error), the format of the Forward_Close response T-PDU shall be of the form shown in Table 194.

Table 194: Forward_Close_Good Response Format

Parameter Name	Format
connection serial number	UINT
vendor ID	UINT
originator serial number	UDINT
application_reply_size; in 16-bit words	USINT
Reserved	USINT
application_reply[]	UINT

A successful Forward_Close response shall be returned when the close has been acknowledged by all the nodes from this point in the path to the end of the path. The response shall indicate the connection_serial_number, vendor_ID, and originator_serial_number of the closed connection. Once the response indicating success has been received, the connection shall be closed

from this point in the path to the end. The target application may optionally pass back application specific information in the successful close response. If no application_reply information is contained in the service the application_reply_size shall be zero.

5.1.1.4.3.3 Format of Forward_Close Response if Failure

If the status parameter of the CM_close_response is not zero (error), the format of the Forward_Close response shall be of the form shown in Table 195.

Table 195: Forward_Close_Bad Response Format

Parameter Name	Format
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size	USINT
reserved	USINT

The object specific status words shall contain information about the reason for the failure. The remaining_path_size shall contain the path size from the point at which the close connection failed.

In the failure response, the remaining_path_size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node shall return either the “pre-stripped” size or 0 for the remaining_path_size.

5.1.1.4.4 Unconnected_Send

5.1.1.4.4.1 Format of Unconnected_Send Request

The Unconnected_Send service shall allow an application to send a message to a device without first setting up a connection. The Unconnected_Send service shall use the Connection Manager object in each intermediate node to forward the message and to remember the return path. The UCMM of each link shall be used to forward the request from Connection Manager to Connection Manager just as it is for the Forward_Open service; however, no connection shall be built. The Unconnected_Send service shall be sent to the local Connection Manager and shall be sent between intermediate nodes. When an intermediate node removes the last port segment, the message shall be formatted as a UCMM message and sent to the port and link address of the last segment.

NOTE The target node never sees the Unconnected_Send service but only a standard message arriving via the UCMM.

If the message contains an odd number of bytes, a pad byte shall be appended to the end of the message allowing the remaining fields to be aligned on a 16-bit boundary. The format of the Unconnected_Send request shall be of the form of the form shown in Table 196.

Table 196: Unconnected_Send Request Format

Parameter Name	Format
priority/tick time	USINT
conn time-out ticks	USINT
Message_size; in bytes, not including pad	UINT
Message Router PDU ; if message_router_PDU is odd size, include pad byte	MR_Request
pad	USINT
path_size	USINT
Reserved	USINT
path[]	UINT

The `message_size` shall be the size of the `message_router_PDU` in bytes without the pad. The reserved field shall be set to zero. The `path_size` shall be the number of 16-bit words in the path field.

5.1.1.4.4.2 Unconnected_Send Response

The `Unconnected_Send` response shall be generated by the last intermediate node from the UCMM response generated by the target node or by an intermediate node as the result of a UCMM time-out, a problem with the embedded message, or a problem with the `Unconnected Service Request` itself. The packet shall be routed from intermediate node to intermediate node using the information stored when the `Unconnected_Send` request was processed. The response shall contain a header with status information about the request and a variable length response generated by the target node.

5.1.1.4.4.3 Format of Unconnected_Send Response if Success

This format shall be used when an unconnected message response is received. The application response shall be the format of the Message response from the target and shall contain error codes that resulted from the execution of the message by the application at the target node. The structure of the response is shown in Table 197.

Table 197: Unconnected_Send_Good Response Format

Parameter Name	Format
<code>application_reply[]</code>	USINT

5.1.1.4.4.4 Format of Unconnected_Send Response if Failure

This format shall be used if the originating node or any intermediate node found a problem in the message format, message parameters, or a UCMM time-out due to no acknowledge or no response received. A UCMM time-out may be the result of

- The target device not being connected or powered up. (link specific time-out);
- An intermediate device not being connected or powered up. (link specific time-out);
- A target device being out of UCMM buffers or overloaded (Flow Control). (link specific time-out);
- An intermediate device being out of UCMM buffers or overloaded (Flow Control). (link specific time-out);
- An invalid path segment type (logical or network segment);
- An invalid port segment address (link or port address invalid).

Since the `Unconnected` message system does not implement any flow control, the failure of an `Unconnected_Send` message need not mean a device is not present in the system. The format of the `Unconnected_Send` response service for a failure shall be of the form shown in Table 198.

Table 198: Unconnected_Send_Bad Response Format

Parameter Name	Format
<code>remaining_path_size</code>	USINT
<code>pad</code>	USINT

In the failure response, the remaining Connection Path Size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining Connection Path Size.

5.1.1.5 CM PDU Components

5.1.1.5.1 Network Connection Parameters

5.1.1.5.1.1 Format

Network connection parameters shall be provided as a single 16-bit word that contains five fields, shown in Figure 42. The fields within the 16-bit word shall indicate

- the type of network connection desired;
- whether the buffer size is variable or fixed;
- the priority of the connection;
- the size of the connection buffer required.

The size of the connection buffer shall include any transport header.

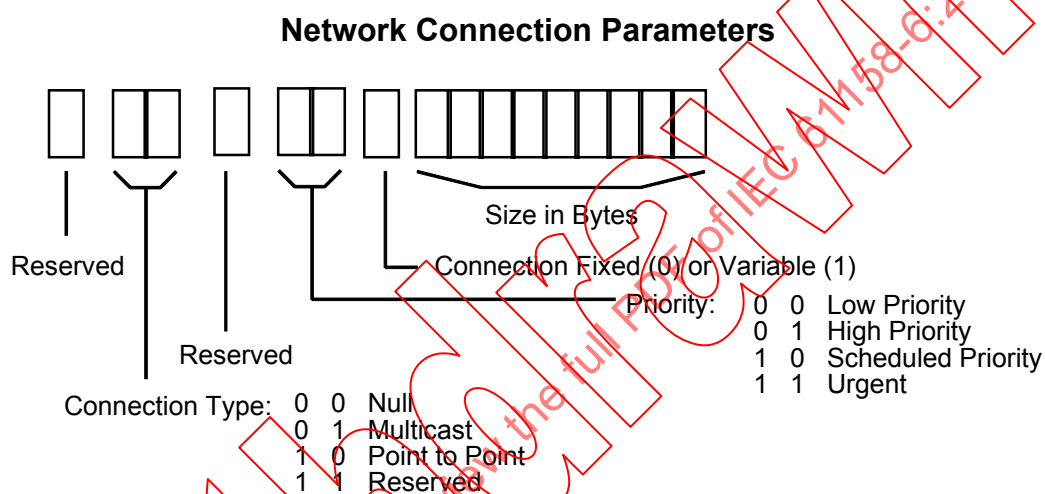


Figure 42: Network connection parameters

5.1.1.5.1.2 Connection Type

The connection type shall be one of NULL, MULTICAST, or POINT2POINT. The NULL type shall indicate that no network connection is required, while MULTICAST and POINT2POINT refer to the network connection types as defined in IEC 61158-5. The NULL connection type may be used to reconfigure a connection.

5.1.1.5.1.3 Priority

Priority shall be one of LOW, HIGH, and SCHEDULED as specified in IEC 61158-4, Data Link Layer Protocol.

NOTE SCHEDULED data is prioritised on a link wide basis; however, the two unscheduled priorities (LOW and HIGH) are arbitrated independently at each node. One node may transmit a LOW priority packet while another node has HIGH priority data to send.

5.1.1.5.1.4 Connection Fixed/variable

The connection can be set up to use fixed or variable sized connections. With a fixed size connection, each transmission on the connection shall use the same size buffer. Otherwise, either a buffer overrun or underrun condition shall occur. With a variable sized connection, each transmission on the connection may send a variable amount of data up to the maximum size, which shall be specified when the connection is opened. Buffer underruns shall not be reported, but a buffer overrun can still occur if too much data is sent.

5.1.1.5.1.5 Connection Size

The network connection size shall be the size of the buffer required for the connection, which includes the data and any transport header. For a variable sized connection, the size shall be the maximum size of the buffer for any transfer. The actual size of the transfer for a variable connection shall be equal to or less than the size specified for the network connection. The maximum buffer size shall be dependent on the links that the connection traverses.

5.1.1.5.2 Reserved Parameters

Reserved bits shall be clear (= 0).

5.1.1.5.3 Packet Interval

5.1.1.5.3.1 Requested Packet Interval (CM_RPI)

The requested packet interval shall be the requested time between packets in microseconds. The format of the CM_RPI shall be a 32-bit integer in microseconds.

5.1.1.5.3.2 Actual Packet Interval (CM_API)

The actual packet interval shall be the actual time between packets in microseconds. The format of the CM_API shall be a 32-bit integer in microseconds.

5.1.1.5.3.3 Usage

The requested packet interval shall be the time between packets requested by the receiving device. The value shall be used to allocate link transmit time at each of the producing nodes. The allocation of link transmit time may have to be adjusted when the actual packet rate or actual packet interval is returned, since it is possible for the two values to differ. The path time-out value at each of the intermediate and target nodes shall also be set to the connection time-out multiplier times the CM_API. The CM_RPI is therefore required for all connections.

5.1.1.5.3.4 Function of Priorities

Scheduled Priority

For scheduled priority, the CM_RPI shall be the packet rate of the repetitive data. On links that support allocation of link transmit time, link transmit time shall be reserved for this packet. For scheduled priority, the data shall also be restricted to the specified packet rate, which means that if data arrives at an intermediate node faster than the specified packet rate, the node shall filter the packets to the specified rate. Since each node's scheduled priority update rate is in discrete quanta, the Actual Packet Interval (CM_API) may be smaller (more rapid) than the CM_RPI. The path time-out value shall be set to the Connection Time-out multiplier times the CM_API.

High Priority

For high priority, the CM_RPI shall be used to set the path time-out in the intermediate and target nodes. The CM_RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the high priority is not quantised at any of the nodes, the CM_API shall equal the CM_RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the CM_API.

Low Priority

For low priority, the CM_RPI shall be used to set the path time-out in the intermediate and target nodes. The CM_RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the low priority is not quantised at any of the nodes, the CM_API shall equal the CM_RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the CM_API.

5.1.1.5.4 Connection Time-out Multiplier

The Connection Time-out Multiplier specifies the multiplier applied to the CM_RPI to obtain the connection time-out value. Device shall stop transmitting on a connection whenever the connection times out even if the pending close has been sent. The multiplier shall be as represented by Table 199.

Table 199 Time-out multiplier

Value	Multiplier	Notes
0	X 4	Default
1	X 8	
2	X 16	
3	X 32	
4	X 64	
5	X 128	
6	X 256	
7	X 512	
8 – 255	reserved	

5.1.1.5.5 Connection Request Priority & Tick time

Two values are packed into this field. The connection request priority determines the priority of the UCMM messages used to establish the connection and has no correlation with the priority of the connection being made. Although Low priority is used for most connection establishments, High priority may be used for those special circumstances which require a connection be made quickly. The use of low priority was chosen to avoid congestion with time-critical messaging. UCMM messages can not use scheduled priority.

The tick time shall be used in conjunction with the connection request time-out ticks value. This value determines the time between "ticks" in the latter field. Thus, if the tick time is 1ms., then a connection request time-out value of 5 would translate into 5 ms. If the tick time was 2 ms, then a connection request time-out value of 5 would translate into 10 ms.

The format of the Connection Request Priority/Tick Time shall be as shown in Figure 43.

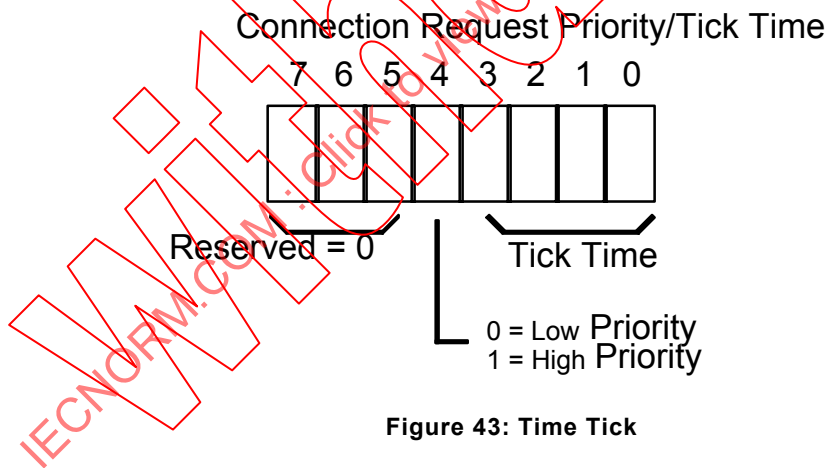


Table 200 shall determine the time between ticks:

Table 200: Time Tick units

Tick Time		
Tick Time (binary)	Time Per Tick	Max Time
0000	1 ms	255 ms
0001	2 ms	510 ms
0010	4 ms	1 020 ms
0011	8 ms	2 040 ms
0100	16 ms	4 080 ms
0101	32 ms	8 160 ms
0110	64 ms	16 320 ms
0111	128 ms	32 640 ms
1000	256 ms	65 280 ms
1001	512 ms	130 560 ms
1010	1 024 ms	261 120 ms
1011	2 048 ms	522 240 ms
1100	4 096 ms	1 044 480 ms
1101	8 192 ms	2 088 960 ms
1110	16 384 ms	4 177 920 ms
1111	32 768 ms	8 355 840 ms

5.1.1.5.6 Connection Request Time-out Ticks

The connection request time-out ticks shall be used to specify the amount of time the originating application shall wait for the connection to be established. Each hop of the connection has a link specific time-out value implemented by the UCMM, and after the automatic retries have been exhausted without an acknowledge the UCMM shall generate a time-out error. Therefore if a device in the path shall be either not present or too busy to acknowledge a UCMM request, a UCMM time-out shall occur as fast as would be possible. The only time the connection request time-out period would be encountered shall be if the target node acknowledges the connection request but fails to respond or if the connection request shall be delivered but a failure in the network prevents the response from being returned. The connection request time-out shall be not the path time-out value derived from the CM_RPI and used to close a connection after it shall be established. The connection request time-out shall be specified by a combination of the connection request time-out ticks multiplied by the time per tick which shall be determined by the tick time value.

Since a particular connection request time-out can be expressed in several different formats, the rule shall be to **always pack it such that the tick time value shall be minimised**. This allows the best granularity for decreasing the time-out as the message is passed through hops. This also means that the tick time value may change as the message is passed through multiple hops. As an example, 300 ms may be encoded as:

1. 0001 10010110 -> Time per Tick = 2 ms, Ticks = 150 (right)
2. 0010 01001011 -> Time per Tick = 4 ms, Ticks = 75 (wrong)

The connection request time-out value shall be dependent on the number of port segments in the connection path, the processing time at the target node, and the congestion of the intermediate networks. The connection request time-out can be a relatively large value since the full connection request time-out shall only occur in a few rare cases. The only time the full connection request time-out would occur would be if:

- the connection request was successfully delivered to the target node and the node failed to generate a response. This would usually be caused by a firmware error in the device.
- the connection request was successfully delivered to the target node and the response was discarded due to congestion at an intermediate node.
- the connection request was successfully delivered to the target node and the target node or one of the intermediate nodes was removed or powered down before the reply was returned.

The UCMM shall be used to forward the connection request to the next node in the path. The UCMM sends the request and then waits a link specific time for an acknowledgement or reply, if no acknowledgement or reply shall be received before the link specific time-out, the request shall be retried. After a link specific number of retries (typically three), the UCMM shall signal a time-out on the connection request. The Connection manager which generated the request shall return an error reply to the connection request. Therefore the connection request shall time-out as soon as possible for the most common system faults

- missing or powered down devices in the path;
- congestion at a device in the path.

To aid in debugging systems, each hop in the path shall decrement the time-out by about 80 ms before sending it out in the Forward_Open to the next device in the path. Each intermediate device (router) shall round the connection time-out value that shall be decremented such that a minimum of 1 tick shall be subtracted. This shall enable the last device before the failure to time-out first and return information about how far the connection was made before the failure. This information can be used to help in debugging problems in a system. A typical system function is shown in Figure 44.

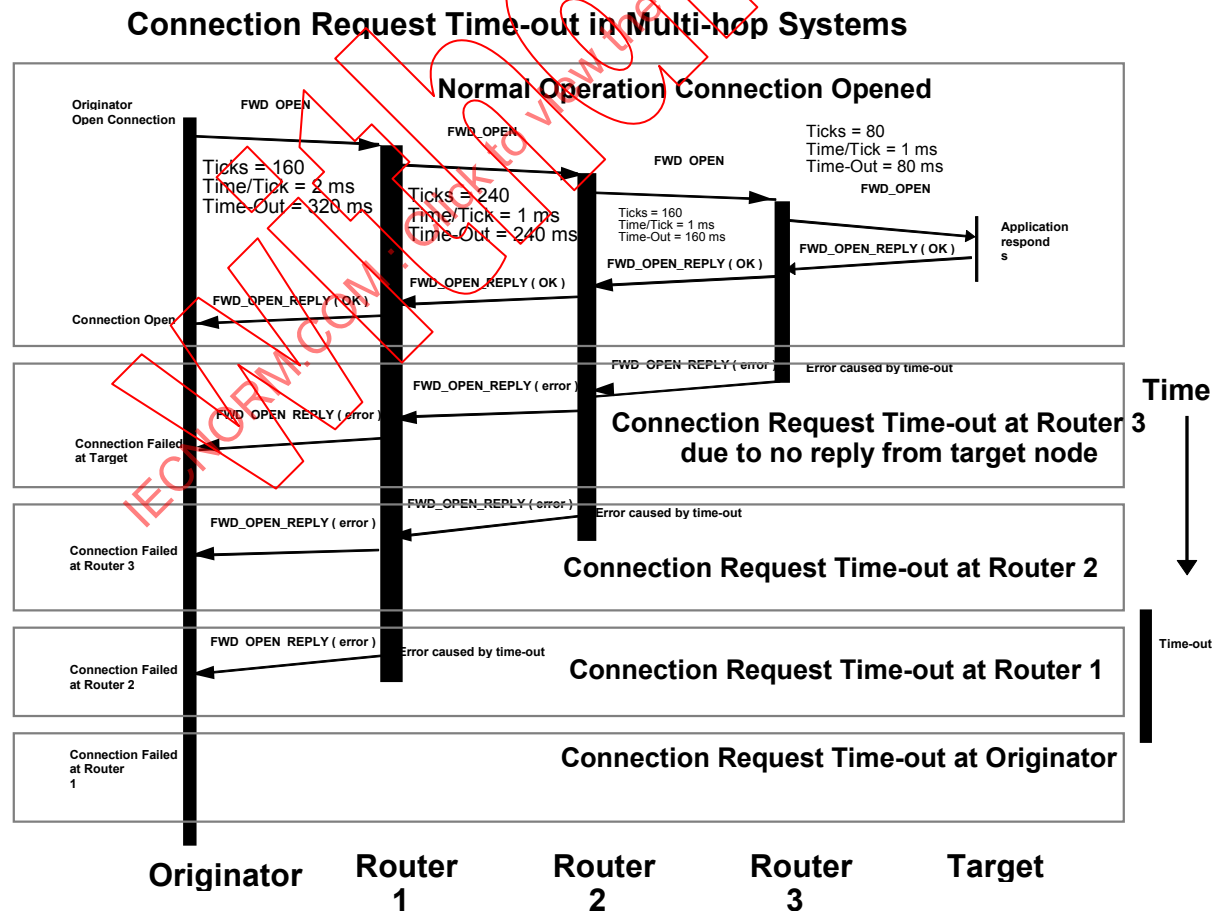


Figure 44: Connection establishment time-out

In Figure 44 a connection is requested over a path containing 4 hops. The Connection request Time-out is specified as 320 ms by the originator of the connection. In the first case the connection is completed and the calculated time-out values are shown at each hop in the path. The second case is where the last device receives an acknowledgement but does not receive a response and times out. It returns a time-out error to the originator. Since the time-out value decreases for each hop, the time-out error shall be returned to the originator before the originator times out.

The other cases show time-outs at the other devices, and in each case the time-out error shall be returned to the originator before the originator itself times out. The last case is where no response is received by the originator and the connection open process is timed out. The reason to decrease the time-out at each node is so additional error information indicating where in the path the time-out occurred can be sent. Remember that if any of the devices do not receive an acknowledgement, the connection request shall immediately be timed out and an error reply shall be generated.

5.1.1.5.7 Connection Serial Number

The connection serial number shall be a unique 16-bit value selected by the connection manager at the originator of the connection. The originator shall make sure that the 16-bit value is unique for the device. There shall be no other significance placed on the number by any other nodes in the connection path. The connection serial numbers shall be unique but do not have to be sequential. For example, an operator interface may have a large number of connections open at the same time, each with a unique number. The same values could be repeated at other operator interface stations. A possible implementation would be to have a connection list which points to the descriptor for each connection, and the connection serial number could be the index into the table.

5.1.1.5.8 Vendor ID

The vendor number shall be a unique number assigned to the various vendors of products. Each vendor has a unique number assigned.

5.1.1.5.9 Originator Serial Number

The originator serial number shall be a unique 32-bit value that is assigned to a device at the time of manufacture. This value shall be guaranteed to be unique for all devices manufactured by the same vendor. No significance shall be attached to the number. The combination of Vendor ID and Originator Serial Number shall be unique throughout the entire system.

5.1.1.5.10 Connection Number

The connection number shall be a 16-bit value that is assigned by the connection manager when a connection is opened. This value allows other nodes to obtain connection data from the connection manager. This number shall not be confused with the Connection Serial Number.

5.1.1.5.11 Connection Path Size

The connection path size shall be the length of the connection path in 16-bit words. The length of the connection path varies during the connection process, since each node in the connection path removes the current port segment and forwards only the remaining path segments to the next node.

5.1.1.5.12 Connection Path

Refer to Path description in 5.1.1.8.

5.1.1.5.13 Connection Remaining Path

Refer to Path description in 5.1.1.8.

5.1.1.5.14 Network Connection ID

The Network Connection ID shall be a 32-bit value built as follows :

- Bits 31-24 shall be reserved and shall be set to zero;
- Bits 23-16 shall contain a node MAC ID depending on selected connection type
- Bits 15-0 shall contain the 16-bit connection number.

Table 201 shows options for selection of Connection ID.

Table 201: Selection of connection ID

Connection Type	MAC ID	Connection Number
Multicast	MAC of Producer	Unique by Device
Point to Point	MAC of Consumer	Unique by Device

The Network Connection ID shall be link specific and shall not be related to the connection serial number, which is connection specific and the same over all the links. The fields of the Network Connection ID shall be used to set the screening mechanism for the specified link.

The generic tag for the Lpackets exchanged over a connection and described in Data Link Layer, IEC 61158-4, shall be built as follows, based on the contents of the CID fields :

- the first transmitted byte of the generic tag shall contain the MAC ID
- the second transmitted byte of the generic tag shall contain the least significant byte of the connection number
- the third transmitted byte of the generic tag shall contain the most significant byte of the connection number.

To guarantee that connection IDs are unique on a link the following rules shall be used when choosing CIDs:

- the producer shall choose the CID for a multicast transport;
- the consumer shall choose the CID for a point-to-point transport;
- on a ControlNet link, the node responsible for the choice of the CID (either producer or consumer) shall insert its MAC ID into bits 23-16 of the CID in the Forward_Open;
- a multicast CID shall not be reused until all connections associated with the CID have been closed or timed out;
- receiving an I'm alive fixed tag packet from a node shall immediately close any connections with that device.

5.1.1.5.15 Transport Class and Trigger

The transport class and trigger specify the type of transport required for the connection. This information shall not be used by the connection manager but passed on to the application. The application shall determine if the transport type is supported and if an instance of the required transport is available. The byte that encode the transport class and trigger shall be of the following form shown in Table 202.

Table 202: Transport Class, Trigger and Is_Server Format

TransportClass (4 bits)	TriggerMode (3 bits)	Is_Server (1 bit)
NULL = 0 DUPLICATE_DETECT = 1 ACKNOWLEDGED = 2 VERIFIED = 3 NONBLOCKING = 4 FRAGMENTING = 5 MULTICAST_FRAG = 6	CYCLIC = 0 CHANGE_OF_STATE = 1 APPLICATION = 2	IS_NOT_SERVER = 0 IS_SERVER = 1

The least significant 4 bits, class, shall determine which transport type is specified and shall be in the range 0 to 6. The class values 7 through 15 shall be reserved. Bits 4-6, called trigger, shall determine what event causes the production of data on the connection and shall be in the range 0 to 2. The trigger values 3 through 7 shall be reserved. For transports that need to specify the target as a server, the is_server field shall be 1; otherwise, it shall be 0. For transport classes 0 and 1, the is_server field shall be ignored. Bit 7 shall be the is_server field;

See IEC 61158-5 for details on the behaviour for the different combinations of `is_server`, `class`, and `trigger`, `CM_RPI`, and `CM_API`.

5.1.1.6 MR Headers

All request packets delivered to the Message Router, either from a connection or from the UCMM, shall have a header of the form shown in Table 203.

Table 203: MR_Request_Header Format

Parameter	Format
Service	USINT
path_size	USINT
path[]	UINT

The first byte, `service`, shall be directly delivered to the application object and shall be in the range 1 through 127. The `path_size` shall determine the number of 16-bit words in the `path` array. The `path` shall contain addressing information for the packet and shall be padded.

Response packets from the Message Router shall have a header of the form shown in Table 204.

Table 204: MR_Response_Header Format

Parameter	Format
Service	USINT
Reserved	USINT
general_status	USINT
Extended_status_size	USINT
Extended_status[]	UINT

The `service` field shall be the value of the `MR_Request_Header.service` field with its most significant bit set. If `MR_Request_Header.service` equals 0x05, the `MR_Response_Header.service` shall equal 0x85. The `general_status` field shall be filled using the `Status Code` parameter of the service response. Likewise the `extended_status`, `extended_status_size`, and `response` shall be filled using the `Extended Status` and other response parameters from the service response, respectively. The `response_size` shall not be explicitly included in the `MR_Response_Header`; it shall be implied by number of bytes in the `MR_Response_Header`.

5.1.1.7 OM_Service_PDU

5.1.1.7.1 General Syntax

5.1.1.7.1.1 Get_Attribute_All

The `Get_Attribute_All_RequestPDU` body is empty.

The `Get_Attribute_All_ResponsePDU` body shall be as specified in Table 205.

Table 205: Structure of Get_Attribute_All_ResponsePDU body

Name	Data type
Attribute Data	Object/class attribute-specific

5.1.1.7.1.2 Set_Attribute_All

The Set_Attribute_All_RequestPDU body shall be as specified in Table 206.

Table 206: Structure of Set_Attribute_All_RequestPDU body

Name	Data type
Attribute Data	Object/class attribute-specific

The Get_Attribute_All_ResponsePDU body is empty.

5.1.1.7.1.3 Get_Attribute_List

The Get_Attribute_List_RequestPDU body shall be as specified in Table 207.

Table 207: Structure of Get_Attribute_List_RequestPDU body

Name	Data type
Attribute Count	UINT
Attribute List	Array of UINT's

The Get_Attribute_List_ResponsePDU body shall be as specified in Table 208.

Table 208: Structure of Get_Attribute_List_ResponsePDU body

Name	Data type	Semantics of values
Attribute Count	UINT	
Attribute Data	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Values 0-255 are reserved to mirror common service status codes. Values 256-65535 are available for object/class attribute-specific errors.
Attribute Value	Object/class attribute-specific	

5.1.1.7.1.4 Set_Attribute_List

The Set_Attribute_List_RequestPDU body shall be as specified in Table 209.

Table 209: Structure of Set_Attribute_List_RequestPDU body

Name	Data type
Attribute Count	UINT
Attribute Data	Array of STRUCT
Attribute Identifier	UINT
Attribute Value	Object/class attribute-specific

The Set_Attribute_List_ResponsePDU body shall be as specified in Table 210.

Table 210: Structure of Set_Attribute_List_ResponsePDU body

Name	Data type	Semantics of values
Attribute Count	UINT	
Attribute Status List	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Values 0-255 are reserved to mirror common service status codes. Values 256-65535 are available for object/class attribute-specific errors.

5.1.1.7.1.5 Reset

The Reset_RequestPDU body shall be as specified in Table 211, if the **optional** "Object Specific Data" service parameter of the Reset request is specified. Else it shall be empty.

Table 211: Structure of Reset_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Reset_ResponsePDU body shall be as specified in Table 212, if the **optional** "Object Specific Data" service parameter of the Reset response is specified. Else it shall be empty.

Table 212: Structure of Reset_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

5.1.1.7.1.6 Start

The Start_RequestPDU body shall be as specified in Table 213, if the **optional** "Object Specific Data" service parameter of the Start request is specified. Else it shall be empty.

Table 213: Structure of Start_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Start_ResponsePDU body shall be as specified in Table 214, if the **optional** "Object Specific Data" service parameter of the Start response is specified. Else it shall be empty.

Table 214: Structure of Start_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

5.1.1.7.1.7 Stop

The Stop_RequestPDU body shall be as specified in Table 215, if the **optional** "Object Specific Data" service parameter of the Stop request is specified. Else it shall be empty.

Table 215: Structure of Stop_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Stop_ResponsePDU body shall be as specified in Table 216, if the **optional** "Object Specific Data" service parameter of the Stop response is specified. Else it shall be empty.

Table 216: Structure of Stop_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

5.1.1.7.1.8 Create

The Create_RequestPDU body shall be as specified in Table 217, if the **optional** "Object Specific Data" service parameter of the Create request is specified. Else it shall be empty.

Table 217: Structure of Create_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Create_ResponsePDU body shall be as specified in Table 218, if the **optional** "Object Specific Data" service parameter of the Create response is specified. Else it shall be empty.

Table 218: Structure of Create_ResponsePDU body

Name	Data type
Instance ID	UINT
Object Specific Data	Object/class service- specific

5.1.1.7.1.9 Delete

The Delete_RequestPDU body shall be as specified in Table 219, if the **optional** "Object Specific Data" service parameter of the Delete request is specified. Else it shall be empty.

Table 219: Structure of Delete_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Delete_ResponsePDU body shall be as specified in Table 220, if the **optional** "Object Specific Data" service parameter of the Delete response is specified. Else it shall be empty.

Table 220: Structure of Delete_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

5.1.1.7.1.10 Get_Attribute_Single

The Get_Attribute_Single_RequestPDU body is empty.

The Get_Attribute_Single_ResponsePDU body shall be as specified in Table 221.

Table 221: Structure of Get_Attribute_Single_ResponsePDU body

Name	Data type
Attribute Data	Object/class attribute-specific

5.1.1.7.1.11 Set_Attribute_Single

The Set_Attribute_Single_RequestPDU body shall be as specified in Table 222.

Table 222: Structure of Set_Attribute_Single_RequestPDU body

Name	Data type
Attribute Data	Object/class attribute-specific

The Set_Attribute_Single_ResponsePDU body shall be as specified in Table 223, if the **optional** "Object Specific Data" service parameter of the Set_Attribute_Single response is specified. Else it shall be empty.

Table 223: Structure of Set_Attribute_Single_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

5.1.1.7.1.12 Find_Next_Object_Instance

The Find_Next_Object_Instance_RequestPDU body shall be as specified in Table 224.

Table 224: Structure of Find_Next_Object_Instance_RequestPDU body

Name	Data type
Maximum Returned Values	USINT

The Find_Next_Object_Instance_ResponsePDU body shall be as specified in Table 225.

Table 225: Structure of Find_Next_Object_Instance_ResponsePDU body

Name	Data type
Number Of List Members	USINT
Instance ID List	Array of UINT

5.1.1.7.2 Identity Object Specific Syntax Elements

5.1.1.7.2.1 Attributes

Class attributes

The format of the Identity Object class attributes shall be as specified Table 226.

Table 226: Identity Object class attributes

Attribute ID	Name	Data type
1	Revision	UINT
2	Max Instance	UINT
6	Max ID Number of Class Attributes	UINT
7	Max ID Number of Instance Attributes	UINT

Instance attributes

The format of the Identity Object instance attributes is as specified in Table 227.

Table 227: Identity Object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Vendor ID	UINT	The value zero shall not be used.
2	Device Type	UINT	A listing of the Device Type ranges can be found in 5.1.1.9.3.
3	Product Code	UINT	The value zero is not valid.
4	Revision	STRUCT of	The value zero shall not be valid for either the Major and Minor Revision fields of a compliant product. A damaged product may return zero to indicate unknown revision if its storage of revision become corrupt.
	Major Revision	USINT	The Major Revision attribute shall be limited to 7 bits. The eighth bit is reserved and shall be zero.
	Minor Revision	USINT	
5	Status	WORD	Bit definitions are described in Table 228.
6	Serial Number	UDINT	
7	Product Name	SHORT_STRING	The maximum number of 8-bit characters in this string shall be 32. Each of the characters shall be in the range 0x20 to 0x7E.

Table 228: Identity Object bit definitions for Status Instance Attribute

Bit(s):	Called:	Definition
0	Owned	1 for TRUE, 0 for FALSE
1		Reserved, set to 0
2	Configured	1 for TRUE, 0 for FALSE
3		Reserved, set to 0
4,5,6,7		See Table 229.
8	Minor Recoverable Fault	1 for TRUE, 0 for FALSE
9	Minor Unrecoverable Fault	1 for TRUE, 0 for FALSE
10	Major Recoverable Fault	1 for TRUE, 0 for FALSE
11	Major Unrecoverable Fault	1 for TRUE, 0 for FALSE
12, 13		Reserved, shall be set to 0
14, 15		Reserved, shall be set to 0

The values of the status bits 4 to 7 shall be as shown in Table 229.

Table 229: Bits 4 – 7 of Status Instance Attribute

Bits 4, 5, 6, 7:	Meaning
0 0 0 0	Self-Testing (power-up) or State Unknown
0 0 0 1	Firmware Update in progress
0 0 1 0	Communication Fault (lost connection)
0 0 1 1	Unkeyed, Awaiting Connection
0 1 0 0	Non-Volatile Configuration Bad
0 1 0 1	Major Fault (see bits 10 and 11 for fault classification and LED states) Minor fault is not a state change
0 1 1 0	Connected, Active
0 1 1 1	Idle (program mode type)
1 0 0 0	reserved, shall not be used
1 0 0 1	
1 0 1 0	reserved for product specific states
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	
1 1 1 1	

5.1.1.7.2.2 Common Services

Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 230.

Table 230: Class level object/service specific response data of Get_Attribute_All

Attribute ID	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Revision (low byte) Default = 1							
	1	Revision (high byte) Default = 0							
2	2	Max Instance (low byte) Default = 1							
	3	Max Instance (high byte) Default = 0							
6	4	Max ID Number of Class Attributes (low byte) Default = 0							
	5	Max ID Number of Class Attributes (high byte) Default = 0							
7	6	Max ID Number of Instance Attributes (low byte) Default = 0							
	7	Max ID Number of Instance Attributes (high byte) Default = 0							

At the **instance level**, the order of the attributes returned in the “Attribute Data” parameter of the Get Attribute All response shall be as defined in Table 231.

Table 231 – Instance level object/service specific response data of Get Attribute All

Attribute ID	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Vendor (low byte)							
	1	Vendor (high byte)							
2	2	Device Type (low byte)							
	3	Device Type (high byte)							
3	4	Product Code (low byte)							
	5	Product Code (high byte)							
4	6	Major Revision							
	7	Minor Revision							
5	8	Status (low byte)							
	9	Status (high byte)							
6	10	Serial Number (low byte)							
	11	Serial Number							
	12	Serial Number							
	13	Serial Number (high byte)							
7	14	Product Name length							
	15	Product Name (1st character)							
	16	Product Name (2nd character)							
	n	Product Name (last character)							

Because the length of the name is not known before issuing the `Get_Attribute_All` service request, implementors shall allow enough memory space to store a response up to 32 characters in length.

To maintain consistency, if any instance attributes are added at some later date, the instance `GetAttribute` All response shall be modified to include the attributes as defined in Table 232.

Table 232: Modified instance level object/service specific response data of Get Attribute All

[illegible]

Reset service

The Reset common service shall have the object-specific parameter specified in Table 233.

Table 233: Object-specific parameter for Reset

Name	Type	Semantics of values
Type	USINT	Value 0 shall emulate as closely as possible cycling power on the item the <i>Identity Object</i> represents. This value shall be the default if this parameter is omitted. Value 1 shall return as closely as possible to the out-of-box configuration, then shall emulate cycling power as closely as possible. Values 2-255 are reserved.

5.1.1.7.3 Message Router Object Specific Syntax Elements

5.1.1.7.3.1 Attributes

Class attributes

The Message Router Object class attributes shall be as specified in Table 234.

Table 234: Message Router Object class attributes

Attribute ID	Name	Data type
1	Revision	UINT
4	Optional attribute list	STRUCT of
	Number of attributes	UINT
	Optional attributes	ARRAY of UINT
5	Optional service list	STRUCT of
	Number services	UINT
	Optional services	ARRAY of UINT
6	Max ID Number of Class Attributes	UINT
7	Max ID Number of Instance Attributes	UINT

Instance attributes

The format of the Message Router Object instance attributes is as specified in Table 235.

Table 235: Message Router Object instance attributes

Attribute ID	Name	Data type
1	Object_List	STRUCT of
	Number	UINT
	Classes	ARRAY of UINT
2	Number available	UINT

5.1.1.7.3.2 Common Services

Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as specified in Table 236. Default values for all unsupported class attributes shall be as shown in Table 236.

Table 236: Class level object/service specific response data of Get_Attribute_All

Class attribute ID	Attribute name, description and default value
1	Revision default = 1
4	Optional attribute list, number of attributes default = 0
5	Optional service list, number of services default = 0
6	Max ID number of class attributes default = 0
7	Max ID number of instance attributes default = 0

Default values for all unsupported instance attributes shall be as shown in Table 237.

Table 237: Instance level object/service specific response data of Get_Attribute_All

Instance attribute ID	Attribute name, description and default value
1	Object list number, number of supported classes default = 0
2	Number available, maximum number of connections default = 0
3	Always = 0x0000

5.1.1.7.4 Assembly Object Specific Syntax Elements

5.1.1.7.4.1 Attributes

Class attributes

The format of the Assembly Object class attributes shall be as specified in Table 238.

Table 238: Assembly Object class attributes

Attribute ID	Name	Data type
1	Revision	UINT
2	Max Instance	UINT

Instance attributes

The format of the Assembly Object instance attributes is as specified in Table 239.

Table 239: Assembly Object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Number of Members in List	UINT	
2	Member List	ARRAY of STRUCT	This attribute has a complex data type.
	Member Data Size	UINT	Size in bits
	Member Path Size	UINT	Size in bytes (0 = Empty Path)
	Member Path	ARRAY of BYTE	Packed path to the member data. See 5.1.1.8 for definition of Path.
3	Data	ARRAY of BYTES	Contain all of the member data packed into one array. This data may contain many different data types. For efficiency it is best to keep this data word aligned by packing it on word boundaries and adding padding as needed. This can be accomplished by using "empty paths" (Member Path Size = 0).
4	Size	UINT	

5.1.1.7.5 Connection Manager Object Specific Syntax Elements

5.1.1.7.5.1 Attributes

Class attributes

The format of the Connection Manager Object class attributes shall be as specified in Table 240.

Table 240: Connection Manager Object class attributes

Attribute ID	Name	Data type
1	Revision	UINT
2	Max Instance	UDINT
4	Optional Attribute List	STRUCT of
	Number attributes	UINT
	Optional attributes	ARRAY of UINT

Instance attributes

The format of the Connection Manager Object instance attributes is as specified in Table 241

Table 241: Connection Manager Object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	OpenReqs	UINT	
2	OpenFormat Rejects	UINT	
3	OpenResource Rejects	UINT	
4	OpenOther Rejects	UINT	
5	CloseReqs	UINT	
6	CloseFormat Rejects	UINT	
7	CloseOther Rejects	UINT	
8	ConnTimeouts	UINT	
9	Connection Entry List	STRUCT of	
	NumConnEntries	UINT	Number of bits used in the ConnOpenBits element. This attribute, divided by 8 and incremented for any remainder, gives the length in BYTES of the array in the ConnOpenBits field.
	ConnOpenBits	ARRAY of BYTE	Bit field. List of connection data. Each bit represents a possible connection. 0 = No Connection. 1 = Connection Established.
11	CpuUtilization	UINT	CPU Utilization in tenths of a percent. Range of 0 - 1000 representing 0 to 100%.
12	MaxBuffSize	UDINT	Amount of buffer space is in Bytes
13	BufSize Remaining	UDINT	Amount of buffer space is in Bytes

5.1.1.8 Message and Connection Paths

5.1.1.8.1 Contents

The path shall specify the object element that is either the target of a connection request, or the destination of a message request. The path shall contain multiple segments which can indicate the route to the next node (in the case of multiple links), what to connect to or where to send a message in

the target device. Each segment shall be comprised of a segment descriptor byte which specifies the segment type and segment information whose size is dependent on the segment type. The two types of path shall be

- padded paths;
- packed paths.

Each segment of a padded path shall be 16 bit word aligned. If a pad byte is required to achieve the alignment, then the segment shall specify the location of the pad byte. A packed path shall not contain the pad bytes.

The possible segment types shall be as follows:

- Port segment (where);
- Logical segment (what);
- Network segment (when or how);
- Symbolic segment;
- Data segment.

5.1.1.8.2 Segment Type

Each segment of the path shall contain a segment type to indicate how the segment is to be interpreted. The segment type is contained in the first byte of the segment. The bits of the first byte of a path segment shall be numbered 0 to 7, where bit 0 shall be the least significant bit of the byte. Bits of the segment type are defined in Figure 45.

Segment type

Segment Type Depends on segment Type

Segment Type			Depends on segment Type			
0	0	0	Port			
0	0	1	Logical			
0	1	0	Network			
0	1	1	Symbolic			
1	0	0	Data / Extended symbolic			
1	0	1	Reserved			
1	1	0	Reserved			
1	1	1	Reserved			

Figure 45: Segment type

5.1.1.8.3 Port Segment

The port segment shall indicate

- communication port through which to leave the node (expressed as a 16-bit number);
- link address of the next device in the path.

Figure 46 shows the overall structure of the port segment.

The bits 5-7 of the first byte shall be zero to indicate the port segment type. Bit 4, the node/slot address size bit, shall be set to 0 if the link address is one byte. Bit 4 shall be set to 1 if the link address is larger than one byte. If the link address is larger than one byte, its size in bytes shall be in the second byte of the port segment.

Bits 0 – 3, the port identifier, shall indicate through which port to leave the node. The port identifier shall specify a port number or an escape to an extended port identifier when the module can support more than 14 ports. Port number 0 shall be reserved. Port number 1 shall only be used to represent a back-plane port. If the port identifier is 15, then a 16-bit field, called the extended port identifier, shall be the next part of the port segment; otherwise, the value of the port identifier shall be the port number.

The port number shall be followed by a link address whose format depends on the port type. If the link address is greater than one byte, then it shall be padded so that the entire port segment is an even number of bytes. The byte specifying the size of the link address shall not include the pad byte. The pad byte shall be set to 0.

NOTE For the common port types, the link address is a single byte. Other port types may require a larger link address.

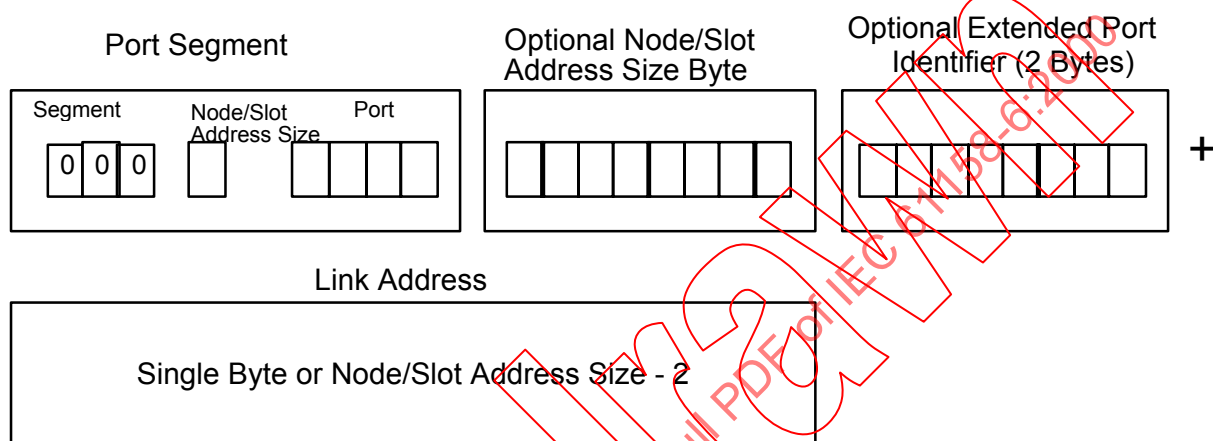


Figure 46: Port Segment

The Extended Port Identifier format of the Port Segment shall only be used when there are more than 14 ports possible on the device. The Port Segment shall always be packed into the smallest Port Segment format possible. Examples of possible port segments are as shown in Table 242.

Table 242: Possible port segment examples

Port Segment	Notes
[02][06]	Port 2, MAC ID 6
[15][0B][31][33][31] [31][35][31][31][33] [32][35][35]	Multi-Byte Address for Port 5, Link Address 13115113255 defined as a character array.
[0F][12][00][01]	Port 18, MAC ID 1

5.1.1.8.4 Logical Segment

The segment type (first byte) of a logical segment shall be in the range 0x20 through 0x3F as shown in Table 243 (the most significant bits shall be 001). The logical segment shall select a particular addressable entity within a device. The segment type shall indicate the format of the logical segment. A packed logical segment shall append a 8-bit, 16-bit or 32-bit integer to the segment type to specify the addressed entity, depending on selected format. A padded 8-bit logical segment shall be identical to a packed 8-bit logical segment. The padded 16-bit logical segments shall insert a pad byte between the segment type and the 16-bit integer, as shown in Table 243. Likewise, the padded 32-bit logical segments shall insert a pad byte between the segment type and the 32-bit integer, as shown in Table 243. The pad byte shall be set to zero.

Table 243: Logical segments

First byte	Logical segment name	Padded format examples (as transmitted)	Packed format examples (as transmitted)	Notes
0x20	8-bit class	[20][04]	[20][04]	class 0x04 (Assembly object)
0x21	16-bit class	[21][00][34][12]	[21][34][12]	class 0x1234
0x22	32-bit class	[22][00][78][56][34][12]	[22][78][56][34][12]	class 0x12345678
0x24	8-bit instance	[24][04]	[24][04]	instance 0x04
0x25	16-bit instance	[25][00][34][12]	[25][34][12]	instance 0x1234
0x26	32-bit instance	[26][00][78][56][34][12]	[26][78][56][34][12]	instance 0x12345678
0x28	8-bit member	[28][04]	[28][04]	member 0x04
0x29	16-bit member	[29][00][34][12]	[29][34][12]	member 0x1234
0x2A	32-bit member	[2A][00][78][56][34][12]	[2A][78][56][34][12]	member 0x12345678
0x2C	8-bit connection point	[2C][04]	[2C][04]	connection point 0x04
0x2D	16-bit connection point	[2D][00][34][12]	[2D][34][12]	connection point 0x1234
0x2E	32-bit connection point	[2E][00][78][56][34][12]	[2E][78][56][34][12]	connection point 0x12345678
0x30	8-bit attribute	[30][04]	[30][04]	attribute 0x04
0x31	16-bit attribute	[31][00][34][12]	[31][34][12]	attribute 0x1234
0x32	32-bit attribute	[32][00][78][56][34][12]	[32][78][56][34][12]	attribute 0x12345678
0x34	electronic key	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	vendor ID = 0x0012 product type = 0xBEEF product code = 0x0B0E major revision = 0x03 minor revision = 0x01 accept compatible keys = yes
0x23, 0x27, 0x2B, 0x2F, 0x33, 0x35 – 0x3F	Reserved			

Class Segment

The class segment shall specify a type of object within a device. For example, the class could specify the Connection Manager (0x06), Message Router (0x02), Assembly (0x04) or Identity object (0x01).

NOTE IEC 61158-5 describes the object model including the definition of class, instance, attribute, connection point, and members of a library of common objects.

Instance Segment

The instance shall be an integer assigned to an object when it is created. The instance segment shall specify the particular instance of an object. Instance 0, called the class instance, shall specify the class itself.

Connection Point Segment

A connection point segment shall be shorthand for a sub-instance of an Assembly object. The connection point segment shall specify the particular connection point.

Attribute Segment

The attributes of an object shall be an enumerated list of externally visible values. An attribute segment shall specify which attribute is selected. These attributes can be a simple data type such as an integer or the attribute may be a complex structure.

Member Segment

Complex attributes shall be composed of members. A member segment shall specify which member of the attribute. For attributes that are arrays, such as instance attribute 0x03 of the Assembly object, the member shall indicate the one-based offset within the array.

Electronic Key Segment

The electronic key segment shall be used to verify the type of target or router device. The key may be included as the first logical segment in a connection path and shall be checked by the Message Router of the receiving node against the values contained in instance 1 of its Identity object before any additional address checks are made. On a multi-hop message, sent to a remote link via intermediate routers, multiple key segments may appear. The format of the electronic key segment shall be as shown in Table 244.

Table 244: Electronic Key Segment Format

Parameter	Format	Value
segment_type	USINT	always 0x34
electronic_key_type	USINT	always 0x04
vendor_ID	UINT	these correspond to the values in instance 1 of the identity object
product_type	UINT	
product_code	UINT	
major_revision : 7	USINT	
compatible_match : 1	USINT	
minor_revision	USINT	

The segment_type for a key segment shall be 0x34. The electronic_key_type shall determine the format of a specific key segment and shall be set to 0x04. All other values for the electronic_key_type shall be reserved. The vendor_ID shall specify the device vendor, or zero if no specific vendor is required.

The product_type shall specify a class of products such as digital input or analogue outputs. The product_type shall be ignored when it is set to zero. The product_code shall be vendor specific with each vendor having a unique code. The product_code shall be ignored when set to zero.

The major_revision and compatible_match fields shall be contained in the least significant 7 bits and most significant bit of the same byte, respectively. The major_revision shall specify the functionality level of a device. The major_revision shall be ignored when set to zero. The minor_revision shall specify the revision of a device that does not effect network-visible behaviour. A value of zero shall specify that the originator of the request accepts any minor revision.

The compatible_match shall modify the key matching function. If the bit is clear (= 0), then any non-zero vendor_ID, product_type, product_code, major_revision, and minor_revision shall match exactly. If the bit is set (= 1), the device may accept the key for any device which it can emulate. The level of emulation shall be product specific.

NOTE IEC 61158-5 describes the Identity object and the rules for updating the revision fields.

5.1.1.8.5 Network Segment

The segment type (first byte) of a network segment shall be in the range 0x40 through 0x5F as shown in Table 245 (the most significant bits shall be 010). The network segment shall be used to specify network parameters which may be required by a node to transmit a message across a network. The network segment shall immediately precede the port segment of the device to which it applies. In other words, the network segment shall be the first item in the path that the device receives.

Table 245: Network segments

First byte	Network segment name
0x40	reserved
0x41	schedule segment
0x42	fixed tag segment
0x43 – 0x5F	reserved

Schedule Segment

The segment type of the schedule network segment shall be 0x41. The schedule network segment shall specify

- a multiplier that, when multiplied by the NUT, gives the CM_API (actual packet interval) of the scheduled transport;
- a phase that determines on which values of the Moderator.interval_count to transmit.

NOTE 1 The Data Link Layer defines the Moderator.interval_count, see IEC 61158-4.

This segment shall be included in the path to a device so that each intermediary node can schedule link transmit time for subsequent scheduled traffic. The multiplier shall be one of 1, 2, 4, 8, 16, 32, 64 or 128. The phase shall be in the range 0 through (multiplier – 1). The second byte of the schedule network segment shall encode both the multiple and phase by adding them together.

NOTE 2 If a transport produces every 64th NUT starting on interval_count = 17, then the encoded value is 17 + 64 = 81.

An encoded value of zero shall specify that the transport has not yet been given permission to use the scheduled priority. If a node receives a request for a scheduled connection in which either no schedule network segment exists, or the schedule network segment exists but the encoded value is 0, the node shall return general status = 0x01, and extended status = 0x0317.

NOTE 3 A connection originator is given permission to use the scheduled priority through the Scheduling object (IEC 61158-5).

Fixed Tag Segment

The segment type of the fixed tag network segment shall be 0x42. The fixed tag network segment shall specify the fixed tag which is to be used when sending an unconnected message. This segment subtype shall precede the port segment within the path. If the fixed tag segment is not present, then the default fixed tag shall be used.

NOTE The fixed tag segment can be used within the path of the Unconnected_Send service (see 5.1.1.4.4) of the Connection Manager to send requests to the Keeper object via the Management UCMM (see IEC 61158-5).

5.1.1.8.6 Symbolic Segment

The range of segment types reserved for symbolic segments shall be 0x60 through 0x7F.

5.1.1.8.7 Data Segment

The segment type (first byte) of a data segment shall be in the range 0x80 through 0x8F (the most significant bits shall be 1000). This IEC 61158-6 only defines the format of the simple data segment with a segment type of 0x80; all other data segment types shall be reserved (0x81 through 0x8F).

Simple Data Segment

The segment type of the simple data segment shall be 0x80. The second byte shall represent the number of 16-bit words of variable length data. Following the second byte shall be the variable length data. A path may contain more than one simple data segment as shown in Table 246.

Table 246: Data Segment

Parameter	Format	Value
segment_type	USINT	always 0x80
segment_size	USINT	size of the data[] array in 16-bit words
data[]	UINT	

The data can be configuration information for an object, additional parameters necessary for the connection, or any other information. The data segment shall not be interpreted by any other device in the path, so only the originating and target applications need agree on its contents.

5.1.1.8.8 Extended Symbol Segment

The segment type (first byte) of an extended symbol segment shall be in the range 0x90 through 0x9F (the most significant bits shall be 1001). This IEC 61158-6 only defines the format of the extended symbol segments 0x91. All other extended symbol segment types shall be reserved (0x90 and 0x92 through 0x9F).

ANSI Extended Symbol Segment

The segment type of the ANSI extended symbol segment shall be 0x91. The second byte shall represent the number of characters (8-bit) in the symbol. Following the second byte shall be the variable length symbol as shown in Table 248.

Table 248: ANSI_Extended_Symbol Segment

Parameter	Format	Value
segment_type	USINT	always 0x91
symbol_size	USINT	size of the symbol[] array
symbol[]	USINT	
pad	USINT	only present if symbol_size is odd, always set to zero

The symbol_size shall be the size of the symbol[] array in bytes and shall not be zero. The symbol_size shall not count the pad byte if it is included.

5.1.1.9 Class, Attribute And Service Codes

5.1.1.9.1 Code Ranges

5.1.1.9.1.1 Defined Ranges

There shall be three categories for address ranges of Class IDs, Attribute IDs and Service Codes as specified in Table 249.

Table 249: Addressing categories

This category	Refers to
Open	A value which has the same meaning for all implementers. All Object Classes and services defined in IEC 61158-5 fall under this category.
Vendor-specific	A range of values specific to the vendor of a device. These are used by vendors to extend their devices beyond the available <i>Open</i> options. A vendor internally manages the use of values within this range. Applies to object classes, attributes, and services.
Object-class specific	A range of values whose meaning is defined by an Object Class. This range applies to Service Code definitions.

The Class ID, Attribute ID and Service code values shall be as defined in Table 250, Table 251 and Table 252.

5.1.1.9.1.2 Class Code ID Ranges

The Class ID values shall be as shown in Table 250. Class Code = 0x00 is reserved and shall not be used.

Table 250: Class code ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00	00	Reserved	1
01 - 63	01 - 99	Open	99
64 - C7	100 - 199	Vendor Specific	100
C8 - EF	200 - 239	Reserved	40
F0 - 2FF	240 - 767	Open	528
300 - 4FF	768 - 1279	Vendor Specific	512
500 - FFFF	1280 - 65535	Reserved	64 256

5.1.1.9.1.3 Attribute ID Ranges

The Attribute ID values shall be as shown in Table 251. Attribute ID = 0x00 is reserved and shall not be used.

Table 251: Attribute ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00	00	Reserved	1
01 - 63	01 - 99	Open	99
64 - C7	100 - 199	Vendor Specific	100
C8 - EF	200 - 239	Reserved	40
F0 - 2FF	240 - 767	Open	528
300 - 4FF	768 - 1279	Vendor Specific	512
500 - FFFF	1280 - 65535	Reserved	64 256

5.1.1.9.1.4 Service Code Ranges

The **Service code** shall be a unique hexadecimal value assigned to each service. The Service Code values shall be as shown in Table 252. Service Code = 0x00 is reserved and shall not be used. The object-specific service codes shall be unique only within the class which define them.

Table 252: Service Code ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00	00	Reserved	1
01 - 31	01 - 49	Open. These are referred to as <i>Common Services</i> . They are defined in Type 2 clauses in IEC 61158-5..	49
32 - 4A	50 - 74	Vendor Specific	25
4B - 63	75 - 99	Object Class Specific	25
64 - 7F	100 - 127	Reserved for future use	28
80 - FF	128 - 255	Reserved for response messages	128

5.1.1.9.2 Code definitions

5.1.1.9.2.1 Communication Object Classes

Table 253 defines the class codes for the Type 2 object classes. All other class codes listed in within the “open” range and not listed in Table 253 shall be reserved. The requirements for these objects are already defined in IEC 61158-5 or will be defined in the future IEC 61158-7.

Table 253: Type 2 Objects

Class code (hex)	Class code (decimal)	Object name
0x01	1	Identity object
0x02	2	Message Router object
0x04	4	Assembly object
0x06	6	Connection Manager object
0xF0	240	ControlNet object
0xF1	241	Keeper object
0xF2	242	Scheduling object

NOTE The ControlNet, Keeper, and Scheduling object classes are part of System Management.

5.1.1.9.2.2 Predefined Class Attributes

Seven predefined Class Attribute IDs shall be reserved, and these predefined/reserved class attributes shall have the definitions listed in Table 254. Because these attributes are reserved, class Attribute ID numbers 1 through 7 shall always be reserved. Therefore, if a class attribute is added to an object class specification, it shall start with Attribute ID #8.

Table 254: Reserved Class Attributes for all object class definitions

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	The starting value assigned to this attribute is one (1). If updates that require an increase in this value are made, then the value of this attribute increases by one (1).
2	Max Instance	UINT or UDINT	The class definition shall define the data type used.
3	Number of Instances	UINT or UDINT	The class definition shall define the data type used.
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Maximum ID Number Class Attributes	UINT	
7	Maximum ID Number Instance Attributes	UINT	

5.1.1.9.2.3 OM_Service codes

Codes of the common services for the deterministic control network objects shall be as shown in Table 255.

NOTE Table 255 lists the codes and names of the common services while IEC 61158-5 provides a general description of each service.

Table 255: Common Services list

Common Service code	Common Service name
0x00	reserved
0x01	Get_Attribute_All
0x02	Set_Attribute_All
0x03	Get_Attribute_List
0x04	Set_Attribute_List
0x05	Reset
0x06	Start
0x07	Stop
0x08	Create
0x09	Delete
0x0A – 0x0D	reserved
0x0E	Get_Attribute_Single
0x0F	reserved
0x10	Set_Attribute_Single
0x11	Find_Next_Object_Instance
0x12 - 0x1B	reserved

5.1.1.9.2.4 CM_Service codes

Codes of the services specific to the Connection Manager shall be as shown in Table 256.

NOTE Table 256 lists the codes and names of the common services while IEC 61158-5 provides a general description of each service.

Table 256: Services specific to Connection Manager

Service code	Common Service name
0x4E	Forward_Close
0x52	Unconnected_Send
0x54	Forward_Open

5.1.1.9.3 Device Types

In order to allow interoperability and interchangeability, a device type shall be used to identify similar devices which

- exhibit the same behaviour;
- produce and/or consume the same set of data;
- contain the same set of configurable attributes.

The formal definition of this information is known as a Device Profile : all devices with the same device type number shall meet the requirements specified in the Device Profile for that device type.

"Device Type" is a required instance attribute of the Identity object as described in IEC 61158-5.

Device types shall be either publicly defined or vendor specific. Table 257 reveals the numbering scheme to be used for device type numbering.

Table 257: Device Type numbering

Range (hex)	Type	Quantity
00 - 63	Publicly Defined - Reserved	100
64 - C7	Vendor Specific	100
C8 - FF	Reserved	56
100 - 2FF	Publicly Defined – Reserved	512
300 - 4FF	Vendor Specific	512
500 - FFFF	Reserved	64256

All publicly defined device types shall have the same meaning for all implementers and are reserved. The Generic Device type (type number = 0x00) shall define a device that does not fit into any of the defined device types.

NOTE The actual definition of publicly defined device types and corresponding Device Profiles is outside the scope of this standard.

Vendor specific device types may be developed and vendors need not publish them. Each vendor shall maintain their own vendor-specific device types and corresponding number allocation.

5.1.1.10 Error Codes

5.1.1.10.1 CM Errors

5.1.1.10.1.1 Error Code Listing

The error codes are returned with the reply to a Connection Manager Service Request which resulted in an error. These error codes shall be used to help diagnose the problem with a Service Request. The error code shall be split into an 8 bit general status and one or more 16 bit words of extended status. Unless specified otherwise, only the first word of extended status shall be required. Additional words of extended status may be used to specify additional module specific debug information. All devices which originate messages shall be able to handle multiple words of extended status.

Table 258 provides a summary of the available error codes.

Table 258: Connection Manager service request error codes

General Status	Extended Status	Explanation
0x00		Service completed successfully.
0x01	0x0100	Connection in Use or Duplicate Forward Open.
0x01	0x0103	Transport Class and Trigger combination not supported
0x01	0x0106	Ownership Conflict
0x01	0x0107	Connection not found at target application.
0x01	0x0108	Invalid Connection Type. Indicates a problem with either the Connection Type or Priority of the Connection.
0x01	0x0109	Invalid Connection Size
0x01	0x0110	Device not configured
0x01	0x0111	CM_RPI not supported. May also indicate problem with connection time-out multiplier.
0x01	0x0113	Connection Manager cannot support any more connections
0x01	0x0114	Either the Vendor Id or the Product Code in the key segment did not match the device
0x01	0x0115	Product Type in the key segment did not match the device
0x01	0x0116	Major or Minor Revision information in the key segment did not match the device
0x01	0x0117	Invalid Connection Point

General Status	Extended Status	Explanation
0x01	0x0118	Invalid Configuration Format
0x01	0x0119	Connection request fails since there is no controlling connection currently open.
0x01	0x011A	Target Application cannot support any more connections
0x01	0x0203	Connection cannot be closed since the connection has timed out
0x01	0x0204	Unconnected Send timed out waiting for a response.
0x01	0x0205	Parameter Error in Unconnected Send Service
0x01	0x0206	Message too large for Unconnected message service
0x01	0x0207	Unconnected acknowledge without reply
0x01	0x0301	No buffer memory available
0x01	0x0302	Network link transmit time not available for data
0x01	0x0303	No Tag filters available
0x01	0x0304	not Configured to send realtime data
0x01	0x0311	Port specified in Port Segment Not Available
0x01	0x0312	Link Address specified in Port Segment Not Available
0x01	0x0315	Invalid Segment Type or Segment Value in Path
0x01	0x0316	Path and Connection not equal in close
0x01	0x0317	Either Segment not present or Encoded Value in Network Segment is invalid.
0x01	0x0318	Link Address to Self Invalid
0x01	0x0319	Resources on Secondary Unavailable
0x01	0x031A	Connection already established.
0x01	0x031B	Connection already established.
0x01	0x031C	Miscellaneous
0x01	0x031D	Reserved
0x02	n/a	Connection Manager resources are unavailable to handle service request
0x03	n/a	Invalid connection number, or specified connection not found.
0x04	Zero Based Word Offset	Segment Type in path is invalid. The Extended Status shall be the word offset (0 based) to the word in the path where the error occurred. The offset starts at the first word after the path size. This error shall not be returned if an error occurs when parsing the Connection Path.
0x05	Zero Based Word Offset	Destination in path is invalid. The Extended Status shall be the word offset (0 based) to the word in the path where the error occurred. The offset starts at the first word after the path size. This error shall not be returned if an error occurs when parsing the Connection Path.
0x07	n/a	Connection has been lost. This is used by the Get/Set Services when they are made through a connection.
0x08	n/a	Connection Manager does not support the requested Service.
0x09	Index to Element	Error in Data Segment. Extended Status shall be index to where the error was encountered in the Data Segment. The Configuration Revision Number if present in the Data Segment shall always be index 1. If the error occurs with the Get/Set Services, then the extended status indicates the attribute number which failed.
0x0C	Optional	Service cannot be performed while Object is in current state. The 1st word of Extended Status may optionally contain the object's current state.
0x10	Optional	Service cannot be performed while Device is in current state. The 1st word of Extended Status may optionally contain the device's current state.
0x11	n/a	Response data too large. This is used by the get services to indicate the amount of data requested was too large to fit into the response buffer.
0x13	n/a	Not enough data was received.

General Status	Extended Status	Explanation
0x14	Attribute Id	Attribute specified in FIND service is not supported by Connection Manager
0x15	n/a	Too much data was received.
0x25	0x0114	Either the Vendor Id or the Product Code in the key segment did not match the device. Used if the Key Segment was contained in the path.
0x25	0x0115	Product Type in the key segment did not match the device. Used if the Key Segment was contained in the path.
0x25	0x0116	Major or Minor Revision information in the key segment did not match the device. Used if the Key Segment was contained in the path.
0x26	n/a	Invalid path size

NOTE 1 The word “n/a” in the Extended Status Column is used to signify that there is no additional Extended Status which is required to be returned for the particular General Status Code.

NOTE 2 The word “optional” in the Extended Status Column is used to signify that if Extended Status information is used, then the first word of that extended status is already defined and user defined extended status shall begin with the second word of extended status.

5.1.1.10.1.2 General Status Code 0x01

Usage

This general status code shall be returned if there is a problem with a parameter in a service request to the connection manager. The following sections shall provide details of the specific extended status codes which are to be returned with the general status.

Connection in Use (Extended Status = 0x0100)

This extended status code shall be returned when an originator is trying to make a connection to a connection point with which the originator may have already established a connection (duplicate Forward_Open). This may result from the originator establishing a connection with a connection point and the success response being lost in the return path. The originator shall then time-out on the request and attempt to establish the connection again even though the target actually established the connection.

The suggested response from the originator in this case shall be to either close and establish the connection again or wait for the connection to time-out and then establish the connection again. It shall be recognised that in the latter case, it shall take the connection 60 seconds (time-out for the first data to be received across a connection) to time-out before the connection can be re-established.

A duplicate forward open shall be defined as an open request which has the same connection serial number, vendor ID, and originator serial number as a connection on the target which is already open. If an intermediate node receives a duplicate Forward_Open request, then the intermediate node shall pass on the request. However, the intermediate node shall not need to allocate transport resources for a new connection.

Transport Trigger Not Supported (Extended Status = 0x0103)

A transport class and trigger combination has been specified which is not supported by the application. Although routers may use the transport/trigger field, they shall not fail the connection based on its contents. Only targets shall return this extended status code.

Ownership Conflict (Extended Status = 0x0106)

The connection cannot be established since another connection already “owns” some of the resources required for this connection. An example of this would be that only one connection can control an output point on an I/O Module. If a second controlling connection is attempted, this error shall be returned. This extended status code shall only be returned by a target node.

Connection Not Found at Target Application (Extended Status = 0x0107)

This extended status code shall be returned by the close connection request, where the connection which is to be closed is not active at the target node. This extended status code shall only be returned by a target node. An intermediate node shall not generate this extended status code. If the specified

connection is not found at the intermediate node, the close request shall still be forwarded using the path specified in the Forward_Close request.

Invalid Connection Type (Extended Status = 0x0108)

This extended status code shall be returned as the result of specifying a connection type (including fixed/variable sized connection) or connection priority which is not supported by the target application. Only a target node shall return this extended status code.

Invalid Connection Size (Extended Status = 0x0109)

This extended status code is returned when the target or router does not support the specified connection size. This could occur at a target because the size does not match the required size for a fixed size connection. It could occur at an intermediate device if the requested size is too large for the specified network.

Device Not Configured (Extended Status = 0x0110)

This extended status code shall be returned when a connection is requested from a target application which has not been configured and the connection request does not contain a data segment for configuration. Only a target node shall return this extended status code.

CM_RPI Not Supported (Extended Status = 0x0111)

This extended status code shall be returned if the device can not support the requested $O \Rightarrow T$ or $T \Rightarrow O$ CM_RPI. This extended status code shall also be used if the connection time-out multiplier produces a time-out value which is not supported by the device.

Connection Manager Out of Connections (Extended Status = 0x0113)

The maximum number of connections allowed by this instance of the Connection Manager has been exceeded.

Vendor ID or Product Code Error (Extended Status = 0x0114)

The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id in the target device.

Product Type Error (Extended Status = 0x0115)

The Product Type specified in the electronic key logical segment does not match the Product Type in the target device.

Revision Mismatch (Extended Status = 0x0116)

The major and minor revision information in conjunction with the exact match bit specified in the electronic key logical segment does not correspond to a valid revision for this device.

Invalid Connection Point (Extended Status = 0x0117)

The connection point specified in the connection path does not correspond to a valid connection point for the target application. This error could also be returned if a connection point was required, but not provided by a connection request.

Invalid Configuration Format (Extended Status = 0x0118)

An instance number specified for the configuration data does not correspond to a configuration instance. For example the connection path specifies a float configuration when the connection points specify a raw connection. This extended status code shall be used when the logical instance segment in the connection path is used to indicate a configuration type instead of an instance.

Controlling Connection Not Open (Extended Status = 0x0119)

The extended status code shall be returned when an attempt is made to establish an echo (i.e. listen only) connection to a connection which has no controlling connection (i.e. owner).

Application Out of Connections (Extended Status = 0x011A)

The maximum number of connections supported by this instance of the Target Application has been exceeded.

For example, the Connection Manager could support 20 connections while the Target Application can only support 10 connections (there may be other Target Applications which an originator can connect to). On the 11th Connection Request to the Target Application, this extended status code would be used to signify that the Target Application is out of Connections.

Connection Timed Out (Extended Status = 0x0203)

This extended status code shall occur when a client tries to send a connected message over a connection that has been timed-out. This extended status code shall only occur at the originating node.

Unconnected_Send Timed Out (Extended Status = 0x0204)

The Unconnected Send Timed Out shall occur when the UCMM times out before a reply is received. This may occur for an Unconnected_Send, Forward_Open, or Forward_Close service. This means that the UCMM has tried a link specific number of times using a link specific retry timer and has not received an acknowledgement or reply. This may be the result of congestion at the destination node or may be the result of a node not being powered up or present. This extended status code shall be returned by the originating node or any intermediate node.

Parameter Error in Unconnected_Send Service (Extended Status = 0x0205)

One of the parameters in the unconnected send service was in error.

This shall be caused by an invalid Connection Tick Time and Connection time-out combination when an intermediate node attempts to decrement this value before passing the Unconnected_Send, Forward_Open, or Forward_Close service on to the next node in the path.

This shall also be caused when the Unconnected_Send is too large to be sent out on a network connection.

Message Too Large for Unconnected Message Service (Extended Status = 0x0206)

The message to be sent via the unconnected message service was larger than fits in the buffers allocated for the unconnected send service. This extended status code can only be detected by the originating node.

Unconnected Acknowledge Without Reply (Extended Status = 0x0207)

The message to be sent via the unconnected message service was acknowledged but not responded to.

No Buffer Memory Available (Extended Status = 0x0301)

The extended status code shall occur when insufficient memory is available to allocate a connection buffer. Routers and target nodes shall detect this error.

Link Transmit Time Not Available (Extended Status = 0x0302)

This extended status code shall be returned by any device in the path which is a producer and can not allocate sufficient link transmit time for the connection on its link. This can occur at any node. This can only occur for connections which are specified as scheduled priority.

No Screeners Available (Extended Status = 0x0303)

Any device in the path which is a consumer and does not have a screener available shall return this extended status code.

Not Configured to Send Real-time Data (Extended Status = 0x0304)

If requested to make a scheduled connection, any device which is unable to send scheduled packets shall return this extended status code. For example, this code shall be returned by a node whose MAC ID is greater than SMAX.

Port Not Available (Extended Status = 0x0311)

This extended status code is the result of specifying a non existent port in a port segment.

Link Address Not Available (Extended Status = 0x0312)

This extended status code is the result of a port segment which specifies an invalid link address. This extended status code shall not be used for valid link addresses which do not respond.

Invalid Segment Type in Path (Extended Status = 0x0315)

This extended status code is the result of a device being unable to decode the connection path. This could be caused by an unrecognised path type, a segment type occurring unexpectedly, or a myriad of other problems.

Error in Close Path (Extended Status = 0x0316)

The path in the close service does not match the connection being closed. This means the connection points to a different module or application than is specified in the path. The connection is deleted but the error message shall be returned.

Scheduling Not Specified (Extended Status = 0x0317)

Either a Schedule Segment was expected in the path and not found, or the Encoded Value in the Schedule Segment was zero (i.e. invalid).

Link Address to Self Invalid (Extended Status = 0x0318)

Under some conditions (depends on the device), a link address in the Port Segment which points to the same device (loopback to yourself) is invalid.

Secondary Resources Unavailable (Extended Status = 0x0319)

In a dual chassis redundant system, a connection request which is made to the primary system shall be duplicated on the secondary system. If the secondary system is unable to duplicate the connection request, then this extended status code shall be returned.

Connection Already Established (Extended Status = 0x031A)

A request for a direct connection has been refused because the connected data is already being sent as a piece of the data sent via another connection.

Connection Already Established (Extended Status = 0x031B)

A request for a connection has been refused because a piece of the connected data is already being sent on a direct connection.

Miscellaneous (Extended Status = 0x031C)

Essentially, if no other extended status code applies, then return this one.

5.1.1.10.1.3 General Status Code 0x02

This general status code shall be returned when the Connection Manager object lacks the resources to handle a connection request.

5.1.1.10.1.4 General Status Code 0x03

This general status code shall be returned by a request when the connection number supplied with the service is invalid (i.e. no connection is present). This general status code is also returned by a service when the connection specified by the Connection Serial Number, Vendor Id, and Originator Serial Number is not found at the target.

5.1.1.10.1.5 General Status Code 0x04

This general status code shall be returned when there is a problem with the Segment Type in the path of a service request. The extended status provides a zero based 16 bit word offset to the place where the error occurred. The offset begins with the first word after the path size in the message.

5.1.1.10.1.6 General Status Code 0x05

This general status code shall be returned when there is a problem with the Destination in the path of a service request. The extended status provides a zero based 16 bit word offset to the place where the error occurred. The offset begins with the first word after the path size in the message.

5.1.1.10.1.7 General Status Code 0x07

The general status code shall be returned by the following services if the requests were made via a message router connection and the connection had been closed:

- Get_Attribute_All;
- Set_Attribute_All;
- Get_Attribute_Single;
- Set_Attribute_Single;
- Get_Attribute_List;
- Set_Attribute_List.

5.1.1.10.1.8 General Status Code 0x08

This general status code shall be returned when the connection manager object on a particular device does not support the requested service.

5.1.1.10.1.9 General Status Code 0x09

This general status code shall be returned when there is an error in the data segment. The extended status shall provide an indication of the problem. When this general status code is returned in response to a Set_Attributes_All request, the extended status shall indicate the attribute number of the first attribute which caused the error.

5.1.1.10.1.10 General Status Code 0x0C

This general status code shall be returned when the state of the target application prevents the service request from being handled. The first word of Extended Status can report the object's current state. For the Connection Manager object, the extended status is considered optional and is not required.

5.1.1.10.1.11 General Status Code 0x10

This general status code shall be returned when the state of the device prevents the service request from being handled. The first word of Extended Status can report the device's current state. For the Connection Manager object, the extended status is considered optional and is not required.

For example, a controller may have a key switch which when set to the "hard run" state causes Service Requests to several different objects to fail (i.e. program edits). This general status code would then be returned.

5.1.1.10.1.12 General Status Code 0x11

This general status code shall be returned by services to indicate that the response buffer is not large enough to hold the data which was requested by the service. This applies in particular to Get_Attribute_All, Get_Attribute_List, and Get_Attribute_Single services.

5.1.1.10.1.13 General Status Code 0x13

This general status code shall be returned when not enough data was provided to perform a service request. This could mean that there wasn't enough data in the Connection Path or the size of the message was too small to even route the message to the appropriate object.

5.1.1.10.1.14 General Status Code 0x14

This general status code shall be returned by the find service when an attribute included in the service is not defined for the Connection Manager. The extended status shall indicate the attribute number of the first attribute which caused the error.

5.1.1.10.1.15 General Status Code 0x15

This general status code shall be returned when too much data was provided to perform a service request. This could mean that there was too much data in the Connection Path or the size of the message was greater than expected for a given service request.

5.1.1.10.1.16 General Status Code 0x25

Usage

This general status code shall be returned if the electronic key logical segment is included as part of the path and a problem was detected with the data contained in the electronic key logical segment. The extended status shall provide details of the error. This general status code shall not be returned if the electronic key logical segment was part of the connection path. In that case, general status 0x01

(Connection Failure) would be returned. This clause provides details of the specific extended status codes which are to be returned.

Vendor ID or Product Code Error 0x0114

The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id in the target device.

Product Type Error 0x0115

The Product Type specified in the electronic key logical segment does not match the Product Type in the target device.

Revision Mismatch 0x0116

The major and minor revision information in conjunction with the exact match bit specified in the electronic key logical segment does not correspond to a valid revision for this device.

5.1.1.10.1.17 General Status Code 0x26

This general status code shall be returned when the path size is invalid. This could mean that not enough or too much data is contained in the path to correctly specify the destination of a service request.

5.1.1.10.2 OM Errors

5.1.1.10.2.1 General Status Format

General status used in the service primitives is specified in Table 259.

Table 259. General status codes

Status code	Name	Description and meaning of status code
0x00	Success	Service was successfully performed by the object specified.
0x01	Connection failure	A connection related service failed along the connection path.
0x02	Resource unavailable	Resources needed for the object to perform the requested behaviour were unavailable. Further object specific information shall be supplied in the object specific status field of the response.
0x03	Invalid value in object specific data parameter of a service request	A portion of the data supplied as a object specific data parameter of a service was invalid. The verification of the data shall be specified in the object specification of the object reporting the error.
0x04	Path segment error	The path segment identifier or the segment syntax was not understood by the processing node. The word offset to the first segment of the path that is not understood shall be supplied in the first word of the object specific status field of the response. The offset shall be zero based and calculated from the first word following the path size in the message. Path processing shall stop when a path segment error is encountered.
0x05	Path destination unknown	The path is referencing an object class, instance or structure element that is not known or is not contained in the processing node. The word offset to the first segment component that references something that is unknown or not present in the processing node shall be supplied in the first word of the object specific status field of the response. The offset shall be zero based and calculated from the first word following the path size in the message. Path processing shall stop when a path destination unknown error is encountered.
0x06	Partial transfer	Only part of the expected data was transferred.
0x07	Connection lost	The messaging connection was lost.
0x08	Unimplemented service	The service requested was not implemented or defined for this class or object instance.
0x09	Invalid attribute value	The value of an attribute of the object or class is invalid. The object specific status shall report the attribute number and the status code of the first attribute refusing data.
0x0A	Attribute list error	An attribute in the Get_Attribute_List or Set_Attribute_List response has a non zero status.

0x0B	Already in requested mode/state	The object is already in the mode/state being requested by the service. The object specific status shall report the current status of the object.
0x0C	Object is not able to perform service in its current mode/state	The object can not perform the requested service in its current mode/state. The object specific status shall report the current status of the object.
0x0D	Object already exists	The requested instance of object to be created already exists.
0x0E	Attribute value not settable	The object attribute is not a settable attribute. The object specific status shall report the number of the attribute refusing data.
0x0F	Access permission does not allow service	The access permissions do not allow the object to perform the service. The access permissions available to the object shall be reported in the extended status. See Table 260 for details concerning extended status.
0x10	Device's mode/state does not allow object to perform service	The device containing the object does not allow the object to perform the service in the current mode/state of the device. The object specific status shall report the device current status, e.g. a controller may have a key switch which when set to the 'hard run' state causes Service Requests to several different objects to fail (i.e. program edits) causing this status code to be returned.
0x11	Reply data too large	The data to be transmitted in the response buffer is larger than the allocated response buffer; therefore, no data was transferred.
0x12	Fragmentation of a primitive value	The service specified an operation that is going to fragment a primitive data value, i.e. half a REAL data type.
0x13	Not enough data	The service did not supply enough data to perform the specified operation.
0x14	Undefined attribute	The attribute specified is not defined for the class or object.
0x15	Too much data	The service supplied more data than was expected (depending on the service and the object, the service might still be processed).
0x16	Object does not exist	The object specified does not exist in the device.
0x17	Service fragmentation sequence not currently in progress	The fragmentation sequence for this service is not currently active for this data.
0x18	No stored attribute data	The attribute data of this object was not saved prior to the requested service.
0x19	Store operation failure	The attribute data of this object was not saved due to some failure during the attempt.
0x1A	Routing failure, request packet too large for network	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service.
0x1B	Routing failure, response packet too large for network	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service.
0x1C	Missing attribute list entry data	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behaviour.
0x1D	Invalid attribute value list	The service is returning the list of attributes supplied with status information for those attributes that were invalid.
0x1E	Embedded service error	An embedded service resulted in an error.
0x1F	Connection Related Failure	A service failed because of an error condition related to the processing of a connection related service. This might occur during connected and unconnected messaging. The same extended status codes used for General Status Code 0x01 are returned for this error's extended status.
0x20	reserved	reserved
0x21	Write once value or medium already written	An attempt was made to write to a write once medium (e.g. WORM drive, PROM) that has already been written, or to modify a value that cannot be changed once established.
0x22	Invalid Reply Received	An invalid reply is received (e.g. reply service code does not match the request service code, or reply message is shorter than the minimum expected reply size). This status code can serve for other causes of invalid replies.
0x23	reserved	reserved

0x24	reserved	reserved
0x25	Key Failure in path	The Key Segment which was included as the first segment in the path does not match the destination module. The object specific status shall indicate which part of the key check failed.
0x26	Path Size Invalid	The size of the path which was sent with the Service Request is either not large enough to allow the Request to be routed to an object or too much routing data was included.
0x27	Unexpected attribute in list	An attempt was made to set an attribute that cannot be set at this time.
0x28	Invalid Member ID	The Member ID specified in the request does not exist in the specified Class/Instance/Attribute.
0x29	Member not settable	A request to modify a non-modifiable member was received.
0x2A - 0xCF	Reserved	Reserved.
0xD0 - 0xFF	Reserved object/class specific	This range of status codes has been reserved for use by object and class specific services.

NOTE IEC 61158-5 contains more detail on the service response general status codes for each common service.

5.1.1.10.2.2 Extended Status Format

The MR_response_Header contains a parameter called ext_stat[] which is labelled extended status data.

Extended status codes used in the service primitives are specified in Table 260.

Table 260: Extended status codes

Code	Format	Meaning	Description
0x04	UINT	path_segment_offset	word offset from the beginning of the path of the incorrect segment
0x05	UINT	path_segment_offset	word offset from the beginning of the path of the first segment that is referencing a non-existent object or class
0x09	UINT	attribute_number	number of the attribute refusing the data value
	UINT	attribute_status	status code explaining reason for invalid attribute value
0x0B	UDINT	object_status	status/state/mode information of the object
0x0C	UDINT	object_status	status/state/mode information of the object
0x10	UDINT	device_status	status/state/mode information of the device
0x19	DWORD	storage_status	status/state/mode information of the storage media
0x1D	UINT	attribute_count	number of attribute values being returned
	STRUCT of	status_of_attributes []	array of status information
	UINT	attribute_number	number of attribute
	UINT	attribute_status	status of attribute

5.1.2 Data Abstract Syntax Specification

5.1.2.1 Transport Format Specification

The lower layers of open system architectures are concerned with the transport of user data among distributed functional units. In these layers, the user data may be regarded simply as a sequence of octets. However, application layer entities may manipulate the values of quite complex data types. To achieve independence between the application layer and lower layers, data types may be specified in an abstract syntax notation.

Supplementing the abstract syntax with one or more algorithms (called encoding rules) may determine the values of the lower layer octets which carry the application layer values. The combination of the abstract syntax with a single set of transfer rules produces a specific transfer syntax.

5.1.2.2 Abstract Syntax Notation

The data type definitions provided in this IEC 61158-6 shall be written in Abstract Syntax Notation One (ASN.1), as defined in ISO/IEC 8824. These type definitions shall be a part of the ASN.1 module “Network DataTypes.” The beginning ASN.1 statement indicating that these definitions are in this module is:

```
control network DataTypes DEFINITIONS ::= BEGIN
```

and the closing ASN.1 statement shall be the keyword “END”.

The abstract definitions that follow shall comprise the set of control network data types. In addition, provision is made to extend or derive new data types based on existing defined types, and to include those in a “type dictionary.”

5.1.2.3 Control Network Data Specification

The notation [typeld] for directly derived, enumerated, subrange and structured bit string data shall mean that the tag shall be taken from the “type” field in the corresponding VariableDictionaryEntry.

```
NetworkData ::= CHOICE{ElementaryData, DerivedData}
ElementaryData ::= CHOICE{
    BOOL,
    FixedLengthInteger,
    FixedLengthReal,
    AnyTime,
    AnyDate,
    AnyString,
    FixedLengthBitString}
DerivedData ::= CHOICE {
    DirectlyDerivedData,
    EnumeratedData,
    SubrangeData,
    StructuredBitStringData,
    ARRAY,
    STRUCT,
    FunctionBlockData}
DirectlyDerivedData ::= [typeId] NetworkData
EnumeratedData ::= [typeId] USINT
SubrangeData ::= [typeId] FixedLengthInteger
StructuredBitStringData ::= [typeId] FixedLengthBitString
FixedLengthInteger ::= CHOICE {SignedInteger,
    UnsignedInteger}
SignedInteger ::= CHOICE {SINT, INT, DINT, LINT}
UnsignedInteger ::= CHOICE {USINT, UINT, UDINT, ULINT}
FixedLengthReal ::= CHOICE {REAL, LREAL}
AnyTime ::= CHOICE {ITIME, TIME, FTIME, LTIME}
AnyDate ::= CHOICE {DATE, TIME_OF_DAY, DATE_AND_TIME}
AnyString ::= CHOICE {STRING, STRING2}
FixedLengthBitString ::= CHOICE {BYTE, WORD, DWORD, LWORD}
BOOL ::= [PRIVATE 1] IMPLICIT BOOLEAN
SINT ::= [PRIVATE 2] IMPLICIT OCTET STRING-- 1 octet
INT ::= [PRIVATE 3] IMPLICIT OCTET STRING-- 2 octets
DINT ::= [PRIVATE 4] IMPLICIT OCTET STRING-- 4 octets
LINT ::= [PRIVATE 5] IMPLICIT OCTET STRING-- 8 octets
```

```

USINT ::= [PRIVATE 6] IMPLICIT OCTET STRING-- 1 octet
UINT  ::= [PRIVATE 7] IMPLICIT OCTET STRING-- 2 octets
UDINT ::= [PRIVATE 8] IMPLICIT OCTET STRING-- 4 octets
ULINT ::= [PRIVATE 9] IMPLICIT OCTET STRING-- 8 octets
REAL  ::= [PRIVATE 10] IMPLICIT OCTET STRING-- 4 octets
LREAL ::= [PRIVATE 11] IMPLICIT OCTET STRING-- 8 octets
STIME ::= [PRIVATE 12] IMPLICIT DINT
DATE  ::= [PRIVATE 13] IMPLICIT UINT
TIME_OF_DAY ::= [PRIVATE 14] IMPLICIT UDINT
DATE_AND_TIME ::= [PRIVATE 15] IMPLICIT SEQUENCE {
    time_of_day  UDINT,
    date        UINT }
STRING ::= [PRIVATE 16] IMPLICIT SEQUENCE {
    charcount    UINT,
    stringcontents OCTET STRING} -- one octet per character
BYTE  ::= [PRIVATE 17] IMPLICIT OCTET STRING-- 1 octet
WORD  ::= [PRIVATE 18] IMPLICIT OCTET STRING-- 2 octets
DWORD ::= [PRIVATE 19] IMPLICIT OCTET STRING-- 4 octets
LWORD ::= [PRIVATE 20] IMPLICIT OCTET STRING-- 8 octets
STRING2 ::= [PRIVATE 21] IMPLICIT SEQUENCE {
    charcount    UINT,
    string2contents OCTET STRING} -- 2 octets/ character
FTIME ::= [PRIVATE 22] IMPLICIT DINT
LTIME ::= [PRIVATE 23] IMPLICIT LINT
ITIME ::= [PRIVATE 24] IMPLICIT INT
STRINGN ::= [PRIVATE 25] IMPLICIT SEQUENCE {
    charsize  UINT,
    charcount  UINT,
    stringNcontents OCTET STRING} -- N octets/ character
SHORT_STRING ::= [PRIVATE 26] IMPLICIT SEQUENCE {
    charcount  USINT,
    stringcontents OCTET STRING} -- one octet per character
ARRAY ::= SEQUENCE OF NetworkData -- All of same base type
STRUCT ::= SEQUENCE OF NetworkData -- May be different types
FunctionBlockData ::= SET{
    inputs    [0] IMPLICIT STRUCT OPTIONAL,
    outputs   [1] IMPLICIT STRUCT OPTIONAL,
    controlInputs [2] IMPLICIT STRUCT OPTIONAL,
    controlOutputs [3] IMPLICIT STRUCT OPTIONAL}

```

5.1.2.4 Data Type Specification / Dictionaries

The definition of an object may include text that defines attributes. Attributes shall be assigned a *Data Type* in an object specification. The Data Type may be one of those defined in this standard or may be an object specific extension to this standard. The following definition shall provide a *Type Specification* for data and shall provide a structure for extending or deriving new data types based on existing defined types.

```

Dictionary ::= CHOICE {VariableDictionary, TypeDictionary}
VariableDictionary ::= SEQUENCE OF VariableDictionaryEntry

```

```

VariableDictionaryEntry ::= SEQUENCE{
    name AnyString,
    id FixedLengthInteger,
    type TypeID,
    ranges SEQUENCE OF Subrange, -- for arrays
    accessPrivilege BOOL {READ_ONLY(0), READ_WRITE(1)}
TypeID ::= OCTET STRING -- ASN.1 encoded tag value of the
-- DataTypeSpecification module
Subrange ::= SEQUENCE {
    minValue FixedLengthInteger,
    maxValue FixedLengthInteger}
TypeDictionary ::= SEQUENCE OF TypeDictionaryEntry
TypeDictionaryEntry ::= SEQUENCE {
    name AnyString,
    type TypeID,
    spec DataTypeSpecification}
DataTypeSpecification ::= CHOICE {
    alt [PRIVATE 0] IMPLICIT AlternateTypeSpec,
    bool [PRIVATE 1] IMPLICIT NULL, -- BOOL
    sint [PRIVATE 2] IMPLICIT NULL, -- SINT
    int [PRIVATE 3] IMPLICIT NULL, -- INT
    dint [PRIVATE 4] IMPLICIT NULL, -- DINT
    lint [PRIVATE 5] IMPLICIT NULL, -- LINT
    usint [PRIVATE 6] IMPLICIT NULL, -- USINT
    uint [PRIVATE 7] IMPLICIT NULL, -- UINT
    udint [PRIVATE 8] IMPLICIT NULL, -- UDINT
    ulint [PRIVATE 9] IMPLICIT NULL, -- ULINT
    real [PRIVATE 10] IMPLICIT NULL, -- REAL
    lreal [PRIVATE 11] IMPLICIT NULL, -- LREAL
    stime [PRIVATE 12] IMPLICIT NULL, -- STIME
    date [PRIVATE 13] IMPLICIT NULL, -- DATE
    tod [PRIVATE 14] IMPLICIT NULL, -- TIME_OF_DAY
    dat [PRIVATE 15] IMPLICIT NULL, -- DATE_AND_TIME
    str1 [PRIVATE 16] IMPLICIT NULL, -- STRING
    byte [PRIVATE 17] IMPLICIT NULL, -- BYTE
    word [PRIVATE 18] IMPLICIT NULL, -- WORD
    dword [PRIVATE 19] IMPLICIT NULL, -- DWORD
    lword [PRIVATE 20] IMPLICIT NULL, -- LWORD
    str2 [PRIVATE 21] IMPLICIT NULL, -- STRING2
    ftime [PRIVATE 22] IMPLICIT NULL, -- FTIME
    ltime [PRIVATE 23] IMPLICIT NULL, -- LTIME
    itime [PRIVATE 24] IMPLICIT NULL, -- ITIME
    strN [PRIVATE 25] IMPLICIT NULL, -- STRINGN
    shstr [PRIVATE 26] IMPLICIT NULL, -- SHORT_STRING
    constructedData CHOICE {
        abbrevStruc [0] IMPLICIT AbbreviatedStrucTypeSpec,
        abbrevArr [1] IMPLICIT AbbreviatedArrayTypeSpec,
        frmlStruc [2] IMPLICIT FormalStrucTypeSpec,
        frmlArr [3] IMPLICIT FormalArrayTypeSpec,
        expBitStr [4] IMPLICIT
ExpandedFixedLenBitStrTypeSpec,
        expStr1 [5] IMPLICIT ExpandedStringTypeSpec,
        expStr2 [6] IMPLICIT ExpandedString2TypeSpec}
    }

```

```

AbbreviatedStrucTypeSpec ::= UINT
AbbreviatedArrayTypeSpec ::= DataTypeSpecification
FormalStrucTypeSpec ::= SEQUENCE OF DataTypeSpecification
FormalArrayTypeSpec ::= SEQUENCE {
    lowBound [0] IMPLICIT FixedLengthInteger, -- Array Lower
Bound
    highBound [1] IMPLICIT FixedLengthInteger, -- Array Upper
Bound
    dataType      DataTypeSpecification }

ExpandedFixedLenBitStrTypeSpec ::= SEQUENCE {
    bitStrType DataTypeSpecification -- BYTE, WORD, DWORD, or
LWORD
    bitFields [7] IMPLICIT BitFieldDef}

BitFieldDef ::= SEQUENCE OF {
    bitDef [2] IMPLICIT OCTET STRING} -- Length is always 2
octets.
-- First octet contains starting
-- Bit Position. Trailing octet
-- contains the number of bits.

ExpandedStringTypeSpec ::= UINT -- String Length In Octets
ExpandedString2TypeSpec ::= UINT -- String Length In Octets
AlternateTypeSpec ::= CHOICE {
    directlyDerivedTypeSpec [0] IMPLICIT      TypeID,
    subrangeTypeSpec [1] IMPLICIT      SubrangeTypeSpec
,
    enumeratedTypeSpec [2] IMPLICIT
EnumeratedTypeSpec,
    fbTypeSpec [3] IMPLICIT      FBTypeSpec}

SubrangeTypeSpec ::= SEQUENCE{
    baseType      TypeID, -- NOTE: minVal and maxVal
minValue      FixedLengthInteger, -- shall be within the
range
maxValue      FixedLengthInteger} -- of baseType values
EnumeratedTypeSpec ::= SEQUENCE OF AnyString
BitNameDefintion ::= SEQUENCE {
    bitName AnyString,
    bitNumber USINT}
FBTypeSpec ::= SET{
    inputs [0] IMPLICIT FbtElementSpec OPTIONAL,
    outputs [1] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlInputs [2] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlOutputs [3] IMPLICIT FbtElementTypeSpec OPTIONAL}

FbtElementTypeSpec ::= SEQUENCE OF ElementSpec
ElementSpec ::= SEQUENCE {
    name AnyString,
    typespec ElementTypeSpec}

```

```

ElementTypeSpec ::= CHOICE {
    [0] IMPLICIT TypeID,
    [1] IMPLICIT SubrangeTypeSpec,
    [2] IMPLICIT EnumeratedTypeSpec,
    [3] IMPLICIT FormalArrayTypeSpec,
    [4] IMPLICIT ExpandedStringTypeSpec,
    [5] IMPLICIT ExpandedString2TypeSpec}
    
```

The following END statement shall terminate the ASN.1 module opened in 5.1.1.

END.

5.2 Type 2 Transfer Syntax

5.2.1 Compact Encoding

5.2.1.1 Encoding Rules

This clause describes the means by which the data types defined in this document, shall be encoded/transferred across the control network. The abstract syntax definition along with a particular set of encoding rules shall result in the transfer syntax. For application user data, a single set of encoding rules shall be defined (Compact Encoding), resulting in the Compact transfer syntax.

Compact Encoding rules shall start with the encoding rules defined in ISO/IEC 8825. Compact Encoding then applies optimisation rules, starting with the outer most Service Data Unit (SDU) and progressing to each successive encapsulated SDU. Compact Encoding shall define a more efficient encoding mechanism by reducing the amount of information (overhead) transferred between devices.

The difference between a Compact encoded value and an ASN.1 encoded value shall be the removal of the fields describing the type and length of the information. The TAG and LENGTH components of an ASN.1-encoded value shall not be transmitted on the control network. In addition, the Compact Encoding rules shall indicate that octet ordering rules are the reverse of those seen in ASN.1.

Given the conditions listed in 5.2.1.2, general rules shall be applied to an ASN.1 encoded value to generate a Compact encoded value. The general rules shall be as follows:

- remove the Identifier Octets;
- remove the "TAG" octets specified by ASN.1;
- remove the Length Octets;
- remove the "LENGTH" octets specified by ASN.1;
- reverse the byte ordering for multiple content octets.

5.2.1.2 Encoding Constraints

The representation of a variable value using Compact Encoding shall be possible with the following restrictions:

- the variable type shall be fixed length and shall have no conditional or optional fields;
- the encoding of a given variable shall be represented with a constant number of octets derived from the type specification of this variable.

5.2.1.3 Examples (Informative)

5.2.1.3.1 BOOL Encoding

The BOOLEAN encoding shall be performed on a single OCTET as described in Table 261.

Table 261: BOOLEAN encoding

If the value is:	Then:
FALSE	bit 0 of the octet is 0 ('00'H)
TRUE	bit 0 of the octet is 1 ('01'H)

A FALSE BOOL shall be represented as shown in Table 262.

Table 262: Example compact encoding of a BOOL value

Octet number	1 st
BOOL	00

5.2.1.3.2 SignedInteger Encoding

Table 263: Encoding of SignedInteger values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
SINT	0LSB							
INT	0LSB	1LSB						
DINT	0LSB	1LSB	2LSB	3LSB				
LINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE to Table 263 The example in Table 264 illustrates the encoding of a variable of type DINT whose value is 0x12345678.

Table 264: Example compact encoding of a SignedInteger value

Octet number	1 st	2 nd	3 rd	4 th
DINT	78	56	34	12

5.2.1.3.3 UnsignedInteger Encoding

Table 265: UnsignedInteger values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
USINT	0LSB							
UINT	0LSB	1LSB						
UDINT	0LSB	1LSB	2LSB	3LSB				
ULINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE to Table 265 Table 266 illustrates the encoding of a variable of type UDINT whose value is 0xAABBCCDD.

Table 266: Example compact encoding of an UnsignedInteger

Octet number	1 st	2 nd	3 rd	4 th
UDINT	DD	CC	BB	AA

5.2.1.3.4 FixedLengthReal Encoding

Table 267: FixedLengthReal values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
REAL	0LSB	1LSB	2LSB	3LSB				
LREAL	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE to Table 267 Table 268 illustrates the encoding of a variable (Float1) whose type is REAL and whose value is Float1: = 10.0. (The assignment of the value is using the IEC 61131-3 notation. The ASN.1 value is {'41200000'H} in IEEE format (1.25×2^3 , exponent is 130(bias 127), fraction is 25)).

Table 268: Example compact encoding of a REAL Value

Octet contents	0LSB	1LSB	2LSB	3LSB
REAL	00	00	20	41

NOTE to Table 267 Table 269 illustrates the encoding of a variable (Float2) whose type is LREAL and whose value is Float2: = –100.0. ({ 'C059000000000000'H} in IEEE format (1.5625×2^6 , exponent is 1029 (bias 1023), fraction is .5625)).

Table 269: Example compact encoding of a LREAL value

Octet contents	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
LREAL	00	00	00	00	00	00	59	C0

5.2.1.3.5 Time Encodings

Table 270 illustrates the encoding of Real numbers with fixedlengths.

Table 270: FixedLengthReal values

Octet Number	1st	2nd	3rd	4th	5th	6th	7th	8th
TIME	0LSB	1LSB	2LSB	3LSB				
DATE	0LSB	1LSB						
TIME_OF_DAY	0LSB	1LSB	2LSB	3LSB				
DATE_AND_TIME	0LSB-Time	1LSB-Time	2LSB-Time	3LSB-Time	0LSB-Date	1LSB-Date		
FTIME	0LSB	1LSB	2LSB	3LSB				
LTIME	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

5.2.1.3.6 String Encodings

This clause gives examples of the Compact Encoding of STRING, STRING2, STRINGN, and SHORT_STRING data values.

NOTE The preferred string type for user supplied string data is STRING2 due to international character string requirements.

Table 271 provides a generic illustration of the encoding of a STRING value.

Table 271: STRING value

	Contents (charcount)		Contents (string contents)
STRING	0LSB	1LSB	0LSB

1 byte character

Table 272 provides a generic illustration of the encoding of a STRING2 value.

Table 272: STRING2 value

	Contents (charcount)		Contents (string2contents)	
STRING2	0LSB	1LSB	0LSB	1LSB

2 byte character

Table 273 provides a generic illustration of the encoding of a STRINGN value.

Table 273: STRINGN value

	Contents (charsize)		Contents (charcount)		Contents (stringNcontents)	
STRINGN	0LSB	1LSB	0LSB	1LSB	0LSB	NLSB

N byte character

Table 274 provides a generic illustration of the encoding of a SHORT_STRING value.

Table 274: SHORT_STRING value

	Contents (charcount)	Contents (short_string)
SHORT_STRING	0LSB	0LSB

1 byte character

Table 275 illustrates the encoding of a string variable whose contents equal "Mill". Encoding examples of all string types are presented. Character coding is specified in ISO/IEC 10646-1. The hexadecimal equivalent is: {'4D696C6C'H} for 8 bit encoding.

Table 275 encodes "Mill" as a STRING type.

Table 275: Example compact encoding of a STRING value

	Contents (charcount)		Contents (string contents)			
STRING	04	00	4D	69	6C	6C

Table 276 encodes "Mill" as a STRING2 type.

Table 276: Example compact encoding of STRING2 value

	Contents (charcount)		Contents (string2 contents)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

2 byte character

Table 277 encodes "Mill" as a SHORT_STRING type.

Table 277: SHORT_STRING type

	Contents (charcount)	Contents (short_string contents)			
SHORT_STRING	04	4D	69	6C	6C

5.2.1.3.7 FixedLengthBitString Encoding

This clause provides examples of the Compact Encoding of BYTE, WORD, DWORD, DWORD, LWORD data values. Figure 47 illustrates the bit placement rules associated with the Compact Encoding of a FixedLengthBitString.

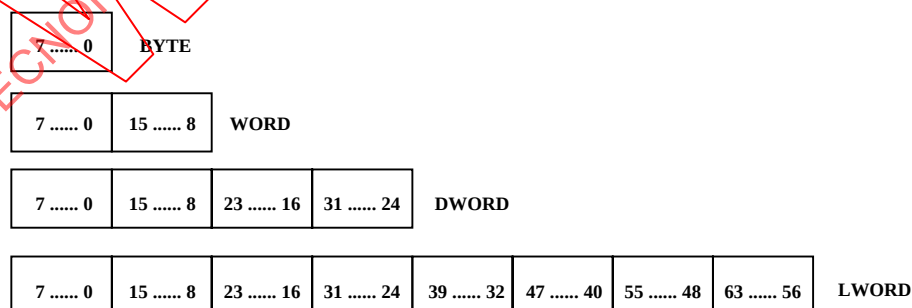


Figure 47: FixedLengthBitString compact encoding bit placement rules

Figure 48, Figure 49, Figure 50 and Figure 51 illustrate the encoding of BYTE, WORD, DWORD, and LWORD.

Bits In Memory: 7 0
00001111

Compact Encoded BYTE
00001111 or 0x0F

Figure 48: Example compact encoding of a BYTE FixedLengthBitString

Bits In Memory: 15 0
00001111 11001111

Compact Encoded WORD
11001111 00001111 or 0xCF0F

Figure 49: Example compact encoding of a WORD FixedLengthBitString

Bits In Memory: 31 0
00001111 11001111 10110110 00111110

Compact Encoded DWORD
00111110 10110110 11001111 00001111 or 0x3EB6CF0F

Figure 50: Example compact encoding of a DWORD FixedLengthBitString

Bits In Memory: 63 0
11110000 11001111 10110110 00111110 11110000 00101101 00011110 00001111

Compact Encoded LWORD
00001111 00011110 00101101 11110000 00111110 10110110 11001111 11110000 or 0x0F1E2DF03EB6CFF0

Figure 51: Example compact encoding of a LWORD FixedLengthBitString

5.2.1.3.8 Array Encoding

5.2.1.3.8.1 One-Dimensional Array Encoding

The Array encoding shall use the encoding rules for the data types for each element and concatenates the elements which compose the array. The encoded values of the array elements shall be encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any data type.

The ASN.1-style definition of a single-dimensional array in the control network shall be:

ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
array_dimension_high_bound, NetworkData }

Assume the following array definition::

ARRAY1 ::= SEQUENCE OF {array_dimension_low_bound := 0,
array_dimension_high_bound := 1, UINT}

Plugging the UINT values {1,2} into this array definition yields the encoding specified in Table 278.

Table 278: Example compact encoding of a single dimensional ARRAY

Octet number	1st	2nd	3rd	4th
ARRAY	01	00	02	00

5.2.1.3.8.2 Two-Dimensional Array Encoding

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound,
    SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound, NetworkData } }
```

5.2.1.3.8.3 Three-Dimensional Array Encoding

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound,
    SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound,
    SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound, NetworkData } } }
```

Since control network data may comprise either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the ARRAY.

A multi-dimensional array shall be seen on the wire as a single-dimensional array. The order of the array elements shall be maintained via the packing/unpacking sequence followed by the end nodes. The sequence followed shall be to access the Nth dimension of the array for all values of the other dimensions.

This shall be achieved by first accessing the array with all dimensions set to their initial index values. After this the Nth dimension is incremented through all of its index values. When the end of the index range for the Nth dimension is reached, the (N-1)th dimension is incremented, and the Nth dimension is set to its initial index value. This process is repeated until all of the array's dimensions have reached the ends of their index ranges, and results in the array being packed into the message buffer as a single-dimensional array. The same procedure is followed to unpack the single-dimensional array into a multi-dimensional array.

The two-dimensional array

```
ARRAY1 [0..1 , 0..2] of UINT := { { 1, 2, 3 },
    { 4, 5, 6 } }
```

results in the data stream shown in Table 279 when it is packed into a single-dimensional array following the compact encoding rules:

Table 279: Example compact encoding of a multi-dimensional ARRAY

Octet number	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th
ARRAY	01	00	02	00	03	00	04	00	05	00	06	00

5.2.1.3.9 Structure Encoding

The structure encoding shall use the encoding rules for the data types for each element and concatenates the elements which compose the structure.

The encoded values of the structure elements shall be encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any Data type.

STRUCT ::= SEQUENCE OF NetworkData — May be different types

Since NetworkData may be comprised of either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the STRUCT.

Assume the following structure definition:

```
newStruct ::= SEQUENCE { BOOL,UINT,DINT}
```

Plugging the values {TRUE,'1234'H,'56789ABC'H} into the structure results in the encoding as specified in Table 280.

Table 280: Example compact encoding of a STRUCTURE

Octet number	1st	2nd	3rd	4th	5th	6th	7th
newStruct	01	34	12	BC	9A	78	56

5.2.1.3.10 Complex Encoded Data Format Examples

The examples 5.2.1.3.10.1 and 5.2.1.3.10.2 show how more complex data formats shall be packed. Example 5.2.1.3.10.1 shows the packing of an array of structures. Example 5.2.1.3.10.2 shows how a structure with an array element is packed.

5.2.1.3.10.1 Example 1: Encoding an Array of Structures:

```
STRUCT1 ::= SEQUENCE OF {
    UINT      ele1;
    USINT     ele2;
    USINT     ele3;
    USINT     ele4;
    UINT      ele5
}

ARRAY1 [ 0..1 , 0..2 ] of STRUCT1 := {
    { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 },
      { 11, 12, 13, 14, 15 } },
    { { 15, 14, 13, 12, 11 }, { 10, 9, 8, 7, 6 },
      { 5, 4, 3, 2, 1 } } }
```

results in the following data stream:

```
[01][00][02][03][04][05][00] [06][00][07][08][09][0A][00]
[0B][00][0C][0D][0E][0F][00] [0F][00][0E][0D][0C][0B][00]
[0A][00][09][08][07][06][00] [05][00][04][03][02][01][00]
```

5.2.1.3.10.2 Example 2: Encoding a Structure With an Array Element

```
STRUCT2 ::= SEQUENCE OF {
    UINT      ele1;
    ARRAY [ 0..2 ] of USINT array2;
    UINT      ele5;
}

STRUCT2 := { 1, { 2, 3, 4 }, 5 }
```

results in the following data stream:

```
[01][00] [02][03][04] [05][00]
```

5.2.2 Data Type Reporting

5.2.2.1 Object Data Representation

Objects may choose to implement a mechanism for reporting Data Type of a particular Attribute or transmitting type information along with the actual data. This clause defines the means by which Data Typing information is conveyed.

The specification of Data Type information utilises the ASN.1 methodology specified in ISO/IEC 8824 and ISO/IEC 8825 with the control network defined optimisations to encode the **DataTypeSpecification** production defined in 5.1.2.4. The control network defined optimisations are:

- The Length Octet of a NULL type is not encoded. For example; the encoding of 'abc [PRIVATE 1] IMPLICIT NULL' would be: 0xC1 (a tag with no length octet).

5.2.2.2 Elementary Data Type Reporting

Elementary data types shall be identified using the identification codes defined in Table 281. These codes shall define the encoding of the primitive members of the DataTypeSpecification production. The control network specifies that ASN.1 NULL types shall not report the Length Octet of zero (0).

Table 281: Identification codes and descriptions of elementary data types

Data type name	Data type code (in hex)	Data type description
BOOL	C1	Logical Boolean with values TRUE and FALSE
SINT	C2	Signed 8-bit integer value
INT	C3	Signed 16-bit integer value
DINT	C4	Signed 32-bit integer value
LINT	C5	Signed 64-bit integer value
USINT	C6	Unsigned 8-bit integer value
UINT	C7	Unsigned 16-bit integer value
UDINT	C8	Unsigned 32-bit integer value
ULINT	C9	Unsigned 64-bit integer value
REAL	CA	32-bit floating point value
LREAL	CB	64-bit floating point value
STIME	CC	Synchronous time information
DATE	CD	Date information
TIME_OF_DAY	CE	Time of day
DATE_AND_TIME	CF	Date and time of day
STRING	D0	character string (1 byte per character)
BYTE	D1	bit string - 8-bits
WORD	D2	bit string - 16-bits
DWORD	D3	bit string - 32-bits
LWORD	D4	bit string - 64-bits
STRING2	D5	character string (2 bytes per character)
ETIME	D6	Duration (high resolution)
LTIME	D7	Duration (long)
ITIME	D8	Duration (short)
STRINGN	D9	character string (N bytes per character)
SHORT_STRING	DA	character sting (1 byte per character, 1 byte length indicator)

5.2.2.3 Constructed Data Type Reporting

5.2.2.3.1 Structure Type Definition

5.2.2.3.1.1 General

The control network defines two different methods for reporting Structure type definitions:

- Formal Encoding (FormalStrucTypeSpec);
- Abbreviated Encoding (AbbreviatedStrucTypeSpec).

Formal encoding shall be used to provide a detailed report of the complete structure definition, including the complete definition of all component data types. Abbreviated encoding shall be used to

specify a *shorter* form of the structure definition. This *shorter* form shall not include the data types associated with the structure's components.

5.2.2.3.1.2 Formal Encoding for Structure Type Information

The examples 5.2.2.3.1.3 and 5.2.2.3.1.4 illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the FormalStrucTypeSpec production defined in 5.1.2.4.

5.2.2.3.1.3 Example 1

Table 282 illustrates the encoding of the following structure definition.

STRUCT ::= SEQUENCE OF { BOOL, UINT, DINT }

Table 282: Example 1 of formal encoding of a structure type specification

STRUCT type	Type length	Component types	Component types	Component types
		BOOL	UINT	DINT
A2	03	C1	C7	C4

NOTE to Table 282 The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.2.3.1.4 Example 2

Figure 52 illustrates the encoding of the following structure definition.

STRUCT_MAIN ::= SEQUENCE OF { UINT, STRUCT_SUB, INT }

with subelement STRUCT_SUB defined as:

STRUCT_SUB ::= SEQUENCE OF { UINT, SINT, INT }

Struct Type	Type Length	Component Types						
		UINT	Struct Type	Type Length	Nested Structure Component Types			
					UINT	SINT	INT	INT
A2	07	C7	A2	03	C7	C2	C3	C3

Figure 52: Example 2 of formal encoding of a structure type specification

5.2.2.3.1.5 Abbreviated Encoding for Structure Type Information

The example 5.2.2.3.1.6 illustrates the abbreviated encoding for structure type specifications. This is actually an example of the encoding of the AbbreviatedStrucTypeSpec production defined in 5.1.2.4.

The UINT defined within the AbbreviatedStrucTypeSpec production shall be initialised with a 16 bit Cyclic Redundancy Check (CRC) value (see Data Link Layer, IEC 61158-4). This can be used by application logic to determine whether or not the format of the structure has changed. The byte stream used to produce the CRC is the actual formally encoded (FormalStrucTypeSpec) structure type specification.

5.2.2.3.1.6 Example

Figure 53 shows the abbreviated encoding of the structure definition presented in 5.2.2.3.1.5:

Struct Type	Type Length	UINT containing CRC	
A0	02	C7	26

Figure 53: Example of abbreviated encoding of a structure type specification

NOTE The algorithms presented in this IEC 61158-6 are used to generate the CRC value of 0x26C7 from the Formally Encoded type specification: [A2][07][C7][A2][03] [C7][C2][C3][C3].

5.2.2.3.2 Array Type Definition

Two different methods for reporting array type definitions shall be:

- Formal Encoding (FormalArrayTypeSpec);
- Abbreviated Encoding (AbbreviatedArrayTypeSpec).

Formal encoding shall be used to provide a detailed report of the complete array definition, including the data content and the array's dimensions. Abbreviated encoding shall be used to specify a *shorter* form of the array definition. This *shorter* form shall not include information specifying the array's dimensions.

5.2.2.3.3 Formal Encoding for Array Type Information

NOTE 1 This clause contains no normative requirements.

NOTE 2 The examples 5.2.2.3.3.1 and 5.2.2.3.3.2 illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the FormalArrayTypeSpec production defined in 5.1.2.4.

5.2.2.3.3.1 Example 1

Figure 54 shows the formal encoding of the following array definition

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 9,
                        UINT
```

Array Type	Type Length	Lower Bound Tag	Lower Bound Length	Lower Bound	Upper Bound Tag	Upper Bound Length	Upper Bound	UINT
A3	07	80	01	00	81	01	09	C7

Figure 54: Example 1 of formal encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.2.3.3.2 Example 2

Figure 55 shows the encoding of the following array definition.

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 19,
                        SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 255,
                        STRUCT_ELE } }
```

in which STRUCT_ELE is defined as:

```
STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }
```

Formal Encoding: [A3][13][80][01][00][81][01][13][A3][0B][80][01][00][81][01][FF][A2][03][C7][C2][C3]

Description:

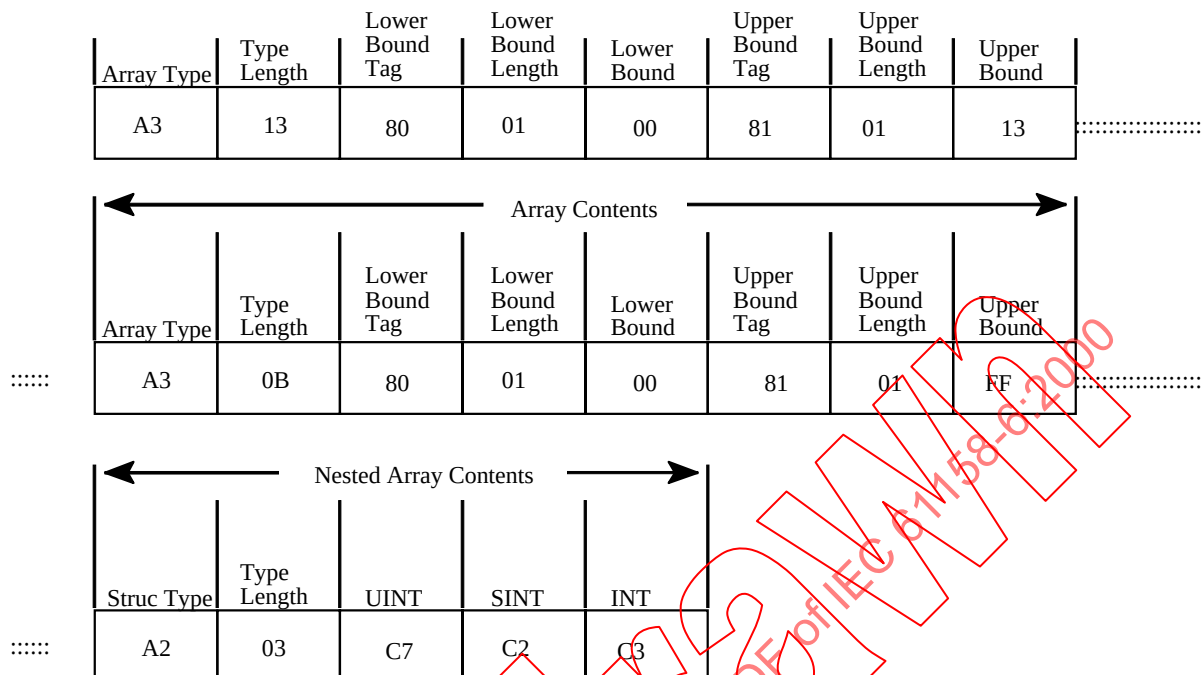


Figure 55: Example 2 of formal encoding of an Array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.2.3.4 Abbreviated Tag Encoding for Array Type Information

NOTE 1 This clause contains no normative references.

NOTE 2 The abbreviated encoding of an Array type shall not include the information specifying the Array's dimensions. This is actually an example of the encoding of the AbbreviatedArrayTypeSpec production defined in 5.1.2.4.

5.2.2.3.4.1 Example 1

Figure 56 shows the abbreviated encoding of the following array definition:

```
ARRAY2 ::= SEQUENCE OF { array_dimension_low_bound := 0,
array_dimension_high_bound := 9,
UINT }
```

Array Type	Type Length	UINT
A1	01	C7

Figure 56: Example 1 of abbreviated encoding of an Array Type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.2.3.4.2 Example 2

Figure 57 shows the abbreviated encoding of the following array definition:

```

ARRAY ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 19,
                        SEQUENCE OF { array_dimension_low_bound := 0,
                                      array_dimension_high_bound := 899,
                                      STRUCT_ELE } }

```

in which STRUCT_ELE is defined as:

```

STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }

```

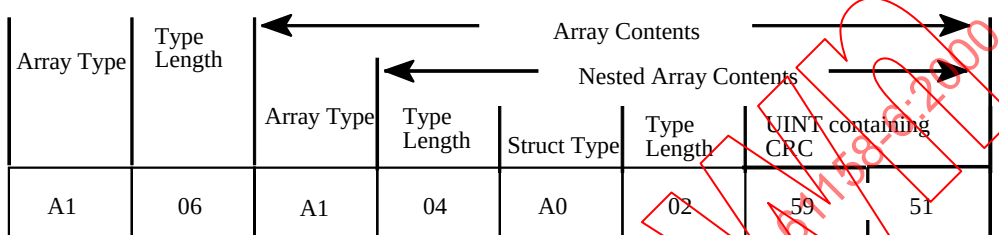


Figure 57: Example 2 of abbreviated encoding of an Array type specification

5.3 Structure of Type 2 FAL Protocol State Machines

Interface to FAL services and protocol machines are specified in this section. Conventions used for the descriptions are given in the generic part of this IEC Standard.

Type 2 fieldbus follows the structure outlined for Type 1 fieldbus with the following specific features :

1. There is no formal definition of AP_Context Machine
2. There is a formally-defined FSPM Machine serving as an interface between FAL User and ARPM.
3. There are ARPM Machines of two different types :
 - a) one ARPM machine for connection-less application relationships
 - b) seven ARPM machines for connection-oriented application relationships
4. DMPM Machine is defined at the interface to the Type 2 Data Link layer

5.4 Type 2 AP Context State Machine

Type 2 fieldbus has no formal AP Context State Machine.

5.5 FAL Service Protocol Machine (FSPM)

Type 2 fieldbus FAL Service Protocol State Machine maps FAL User services onto services of communication objects internal to Type 2 FAL.

5.5.1 Primitive Definitions

The Objects within FSPM shall provide the following services :

Get_Attribute_All Reads values of all attributes of the specified object class or instance

Set_Attribute_All Writes specified values to all attributes of the specified object class or instance

Get_Attribute_List Reads values of the specified list of attributes of the specified object class or instance

Set_Attribute_List Writes specified values to the specified list of attributes of the specified object class or instance

Get_Attribute_Single Reads value of the specified attribute of the specified object class or instance

Set_Attribute_Single Writes specified value to the specified attribute of the specified object class or instance

Reset Resets the specified object class or instance

Create Creates an instance of the specified object class

Delete Deletes an instance of the specified object class

Start Starts execution of the specified object.

Stop Stops execution of the specified object.

Find_Next_Object_Instance Finds the identifier of the next unused instance of the specified object class.

Refer to IEC 1158-5 for detailed definition of these services.

Primitives of Common Services exchanged between UCMM and FAL User are shown in the following Table 283 and Table 284.

NOTE All services have the following common parameters :

Common parameters		Req	Ind	Rsp	Cnf
Argument		M			
AREP		M			
Local		S			
UCMM Record identifier		S			
Transport identifier		S			
Receiver/Server Local ID			M		
Path		M	C		
Routing Path		M			
Add. Path		U	U(=)		
Port ID			M		
Result (+)				S	S(=)
AREP					M=
Receiver/Server Local ID				M=	
Service status				M	M(=)
Result (-)				S	S(=)
AREP					M=
Receiver/Server Local ID				M=	
Service status				M	M(=)

Only those argument parameters additional to the common ones are shown in Table 283 and Table 284.

Additional parameters of indication service primitives are the same as those of the corresponding request primitive.

Additional parameters of confirmation service primitives are the same as those of the corresponding response primitive.

Table 283: Primitives issued by FAL User to FSPM

Primitive name	Source	Additional parameters	Functions
Get_attribute_all.req	FAL User	None	Conveys a request from FAL User to a target object to supply values of all attributes of the specified object class or instance
Set_attribute_all.req	FAL User	Attribute Data	Conveys a request from FAL User to a target object to write specified values of all attributes of the specified object class or instance
Get_attribute_list.req	FAL User	Attribute Count Attribute List	Conveys a request from FAL User to a target object to supply values of specified list of attributes of the specified object class or instance
Set_attribute_list.req	FAL User	Attribute Count Attribute Data	Conveys a request from FAL User to a target object to write specified values of specified list of attributes of the specified object class or instance
Get_attribute_single.req	FAL User	None	Conveys a request from FAL User to a target object to supply values of the specified attribute of the specified object class or instance
Set_attribute_single.req	FAL User	Attribute Data	Conveys a request from FAL User to a target object to write the specified values into specified attribute of the specified object class or instance
Reset.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to reset specified target object class or instance
Create.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to create an instance of the specified object class
Delete.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to delete an instance of the specified object class
Start.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to start an instance of the specified object class
Stop.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to stop an instance of the specified object class
Find_next_object_instance.req	FAL User	Maximum Returned Values	Conveys a request from FAL User to a target device to find the identifier of the next unused instance of the specified object class
Get_attribute_all.rsp	FAL User	(+) Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of all attributes of the specified object class or instance
Set_attribute_all.rsp	FAL User	(+) None (-) Specific Status Code	Returns a response from FAL User to a target object to write specified values of all attributes of the specified object class or instance
Get_attribute_list.rsp	FAL User	(+) Attribute Count Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of specified list of attributes of the specified object class or instance
Set_attribute_list.rsp	FAL User	(+) Attribute Count Attribute Status List (-) Specific Status Code	Returns a response from FAL User to a target object to write specified values of specified list of attributes of the specified object class or instance
Get_attribute_single.rsp	FAL User	(+) Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of the specified attribute of the specified object class or instance
Set_attribute_single.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to write the specified values into specified attribute of the specified object class or instance
Reset.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the an instance of the specified object class has been reset
Create.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the an instance of the specified object class has been created
Delete.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the an instance of the specified object class has been deleted
Start.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the an instance of the specified object class has started
Stop.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the an instance of the specified object class has stopped
Find_next_object_instance.rsp	FAL User	(+) Number Of List Members Instance ID List (-) Specific Status Code	Returns a response from FAL User to a target device to find the identifier of the next unused instance of the specified object class

NOTE Start and Stop services are available only for connection-less (UCMM) transports.

Table 284: Primitives issued by FSPM to FAL User

Primitive name	Source	Additional parameters	Functions
Get_attribute_all.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_all.req
Set_attribute_all.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_all.req
Get_attribute_list.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_list.req
Set_attribute_list.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_list.req
Get_attribute_single.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_single.req
Set_attribute_single.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_single.req
Reset.ind	FSPM	same as in .req primitive	Indicates reception of Reset.req
Create.ind	FSPM	same as in .req primitive	Indicates reception of Create.req
Delete.ind	FSPM	same as in .req primitive	Indicates reception of Delete.req
Start.ind	FSPM	same as in .req primitive	Indicates reception of Start.req
Stop.ind	FSPM	same as in .req primitive	Indicates reception of Stop.req
Find_next_object_instance.ind	FSPM	same as in .req primitive	Indicates reception of Find_next_object_instance.req
Get_attribute_all.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_all.rsp
Set_attribute_all.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_all.rsp
Get_attribute_list.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_list.rsp
Set_attribute_list.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_list.rsp
Get_attribute_single.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_single.rsp
Set_attribute_single.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_single.rsp
Reset.cnf	FSPM	same as in .rsp primitive	Indicates reception of Reset.rsp
Create.cnf	FSPM	same as in .rsp primitive	Indicates reception of Create.rsp
Delete.cnf	FSPM	same as in .rsp primitive	Indicates reception of Delete.rsp
Start.cnf	FSPM	same as in .rsp primitive	Indicates reception of Start.rsp
Stop.cnf	FSPM	same as in .rsp primitive	Indicates reception of Stop.rsp
Find_next_object_instance.cnf	FSPM	same as in .rsp primitive	Indicates reception of Find_next_object_instance.rsp

5.5.2 Parameters of Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 285.

Table 285: Parameters used with Primitives Exchanged between FAL User and FSPM

Parameter Name	Description
AREP : Local identifier UCMM Record identifier Transport identifier	Identifies the entity to be used to convey the service. This parameter may use a dedicated identifier associated with a local entity, the identifier of a UCMM transaction record previously created, or the transport identifier returned by the connection establishment process and associated with the selected AR.
Receiver/Server Local ID	Generated by the Message Router ASE of the responding node. Identifies locally this invocation of the service. It is used to associate service responses with indications.
Path : Routing Path Additional path	In the request, it specifies the FAL APO or FAL APO element that is the destination of the service request. In the indication, it contains only those segments beyond the logical class segment from the service request, i.e. the optional additional information for the target APO (Add.Path).
Port ID	Indicates on which port of the device the service indication arrived.
Service status : Status Code (mandatory) Extended Status (conditional)	Provides information on the result of service execution. It is returned in all confirmed service response primitives (+ and -). Status code indicates the type of error (see IEC 61158-5 for details). Extended status code gives details of the status (see IEC 61158-5 for details).
Attribute Data	1) A stream of information containing all of the attributes. Classes/Objects which support this service shall define the format of this parameter. 2) An array of structures, predefined by the system for the given service.
Attribute Count	Number of attribute numbers in the attribute list
Attribute list []	List (array) of the attribute numbers of the attributes to get from the class or object
Attribute Number	Number representing the attribute value
Attribute Status	Status information of attribute
Attribute Value	Sequence of data specific to the attribute of the object or class
Object Specific Data	Class/Instance specific parameters. Class/Instance specification shall specify the format.
Instance ID	Value assigned to identify the newly created object
Maximum Returned Values	Maximum number of Instance ID values to be returned in the response message
Number Of List Members	Number of Instance IDs specified in response message
Instance ID List	Returned Instance ID List. The Instance IDs are returned in 16 bit integer fields.

5.5.3 FSPM State Machines

FSPM State Machine has got only one possible state : Running.

5.6 Application Relationship Protocol Machines (ARPMs)

Type 2 fieldbus has Application Relationship Protocol Machines for :

- connection-less application relationships
- connection-oriented application relationships

Connection-less relationship is of one type only.

Connection-oriented relationships fit into one of 7 transport classes :

- 0 Null (or Base)
- 1 Duplicate Detection
- 2 Acknowledged
- 3 Verified
- 4 Non-blocking
- 5 Non-blocking, Fragmenting
- 6 Multicast, Fragmenting

Although similar, each of these transport classes has its own ARPM.

5.6.1 Connection-less ARPM (UCMM)

Functions of the connection-less ARPM are provided by the Unconnected Message Manager (UCMM) object. UCMM shall provide an unconnected request/response message services, limited to a single

link that supports multiple outstanding messages. The required number of outstanding messages shall be implementation specific. The UCMM shall be present in all nodes, and shall support at least one outstanding message.

5.6.1.1 Primitive Definitions

The UCMM Object shall provide the following services :

- UCMM_Create Creates an instance of transaction record. Puts UCMM into Running state. This service is local, it does not result in a PDU being sent.
- UCMM_Delete Deletes existing transaction record. Puts UCMM into Inactive state. This service is local, it does not result in a PDU being sent. This service is local, it does not result in a PDU being sent.
- UCMM_Write Writes an item of application data into Transport PDU buffer ; this results in sending the data to a specified target.
- UCMM_Abort Aborts existing transaction.

Refer to IEC 1158-5 for detailed definition of these services.

Primitives exchanged between UCMM and FSPM are shown in the following Table 286 and Table 287

Table 286: Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
UCMM_Create_req	FSPM	fixed tag max retries,	Conveys a request from the FSPM to the ARPM to create a transaction record.
UCMM_Delete_req	FSPM	record	Conveys a request from the FSPM to the ARPM to delete previously established transaction record.
UCMM_Write_req	FSPM	record destination ID UCMM service response timeout transaction priority application data	Conveys a request from the FSPM to the ARPM to send an item of application data using a previously created transaction record.
UCMM_Write_resp	FSPM	record application data	Conveys a response from the FSPM, via Message Router to the ARPM to send a response to UCMM_Send_req using a previously created transaction record.
UCMM_Abort_req	FSPM	record	Aborts the transaction corresponding to the specified transaction record ; removes the packet from the local DLL if it has not yet been transmitted.

Table 287: Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
UCMM_Create_conf	ARPM	record service_status	Conveys a confirmation from the ARPM to the FSPM that transaction record has been created.
UCMM_Write_ind	ARPM	record source ID application data	Conveys an indication from the ARPM to the Message Router that data has arrived on the previously created transaction record. The Message Router then passes the indication to the appropriate application object.
UCMM_Write_conf	ARPM	record service status application data	Conveys a confirmation from the ARPM to the Message Router of the execution of FSPM UCMM_Send_req on a previously created transaction record.

5.6.1.2 Parameters of Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 288 .

Table 288: Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
record	Identifies unambiguously the instance of the temporary connection along which the FSPM has issued a UCMM_Send request primitive. A means for such identification is not specified in this document.
application data	Conveys FAL-User data.
fixed tag	Set to value of 0x83 for UCMM and 0x88 for Management UCMM
destination ID	Unambiguous identifier (MAC ID) of the source node of UCMM_write_req.
source ID	Unambiguous identifier (MAC ID) of the destination node of UCMM_write_req.
service	The service parameter shall specify the delivery mechanism to use for this message and shall be one of : Request_With_Acknowledge = 2 (retries untill acknowledged) Request_With_No_Response = 4 (no acknowledgement) Request_With_No_ACK = 7 (retries untill response) Request_With_No_Retry = 8 (with response) NOTE: These values correspond to command codes contained in the UCMM header
timeout	Duration of the transaction in microseconds. If no UCMM_Send_conf is received before the time-out expires, the transaction is aborted and the send_status parameter shall be TIMEOUT.
retries	Conveys the maximum allowable number of retries.
create_status	Status returned with UCMM_Create_conf = - success - cannot create
send_status	Status returned with UCMM_Send_conf = - success - record_not_created - packet_too_big - no_acknowledge - aborted - timeout

5.6.1.3 UCMM State Machines

5.6.1.3.1 UCMM Client

5.6.1.3.1.1 States

The defined states and their descriptions of the UCMM Client are listed in Table 289 below :

Table 289: UCMM Client States

State	Description
Inactive	The record for UCMM transfer does not exist.
Running	The record for UCMM transfer has been created.
Waiting for Response	Client sent an item of application data and is waiting for a response.

5.6.1.3.1.2 State Transition Diagram

Figure 58 shows the state transition diagram of UCMM client.

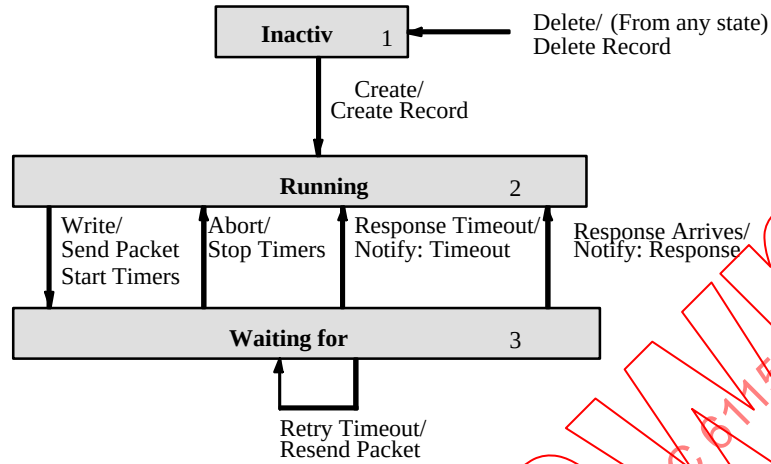


Figure 58: State transition diagram of UCMM Client

5.6.1.3.1.3 State Event Matrix

The state transitions for the UCMM Client shall be as specified in Table 290.

Table 290: State event matrix of UCMM Client

Event	State		
	Inactive	Running	Waiting for Response
Create.req	• create record • transition to Running		
Delete.req	no action	• delete record • increment SEQUENCE • transition to Inactive	• Delete record • increment SEQUENCE • transition to Inactive
Write.req	confirm status = INVALID_RECORD	• send packet • start Response Timer • initialise Retry Count • start Retry Timer • transition to Waiting	confirm status = BUSY
Abort.req	confirm status = INVALID_RECORD	no action	• stop timers • increment SEQUENCE • transition to Running
Retry Timeout Request packet available			1) Decrement Retry Count IF Retry Count expired 2) Notify: Timeout-No ACK 3) Free request buffer 4) Increment Sequence # 5) Stop Response Timer 6) Transition to Running ELSE. 1) Resend packet 2) Restart Retry Timer
Response Timeout			• confirm with status = TIMEOUT-No Response • increment SEQUENCE • transition to Running
Response Arrives		no action	• confirm with status = SUCCESS • Send ACK_RESP, unless response indicates no ACK_RESP desired • increment SEQUENCE • transition to Running
ACK_REQ Arrives, sequence number matches stored value		No Action	• Free request buffer • stop retry timer
ACK_REQ Arrives, sequence number not equal to stored value		No Action	No Action

The sequence number shall be set to zero at initialisation. The sequence number value shall be maintained in the inactive state, to be used when the record transitions to running. The retry timeout value shall be fixed for the link at a value which guarantees that both the client and server nodes have an opportunity to transmit an unscheduled packet.

The response time-out is the response time provided when the record is created.

5.6.1.3.2 High-end UCMM Server

5.6.1.3.2.1 Functions

When a packet arrives at the server an instance of a transaction shall be created if resources are available. If resources are not available the packet shall be dropped. When the instance is created, a transaction record shall be created for that instance. This record shall be active for the life of this transaction. The transaction shall end :

- when an ACK_RESP is received after a response was sent;
- when the response time timer expires;
- or, when a new packet is received after a response was sent.

5.6.1.3.2.2 States

The defined states and their descriptions of the UCMM High-end Server are listed in Table 291 below :

Table 291: High-end UCMM Server States

State	Description
Inactive	The record for UCMM transfer does not exist.
Waiting for Response	Data packet arrived with new Transaction ID. New record is created. The Server is waiting for response to be sent by the Server application.
Response sent	Server application sent a response

5.6.1.3.2.3 State Transitions

Figure 59 shows the high-end UCMM Server state transition diagram.

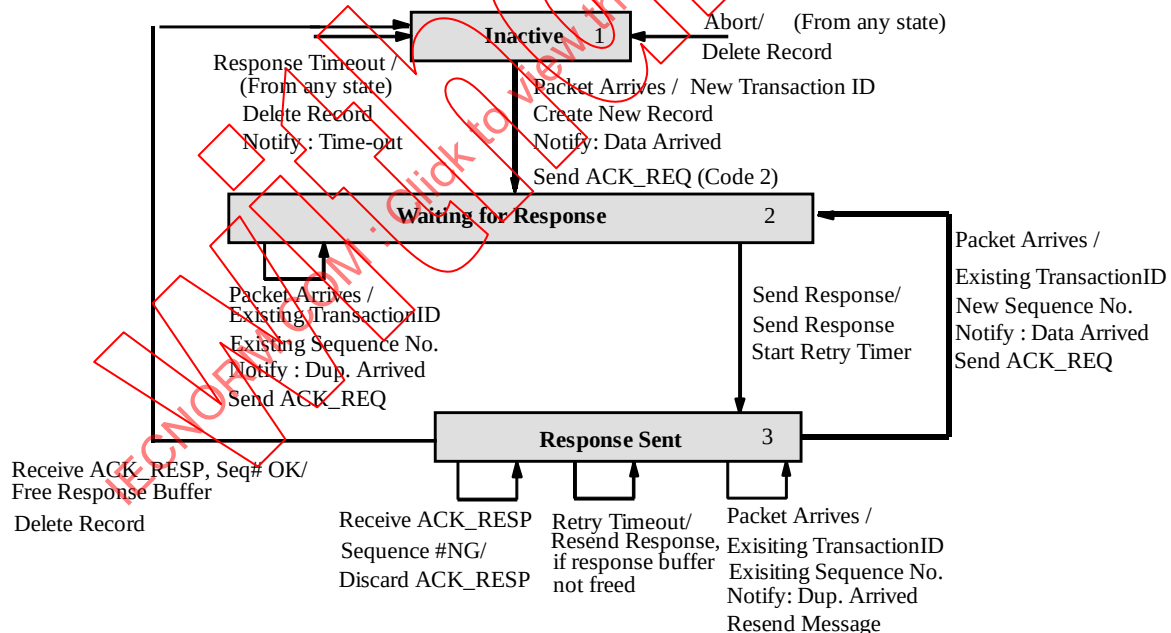


Figure 59: State transition diagram of High-end UCMM server

5.6.1.3.2.4 State Event Matrix

The state transitions for the high-end UCMM server shall be as specified in Table 292.

The sequence number shall be set to zero at initialisation. The sequence number value shall be maintained in the Inactive state, to be used when the record transitions to Running. The retry time-out value shall be fixed for the local link.

The response time-out shall be the response time provided when the record is created.

Table 292: State event matrix of High -end UCMM server

Event	State		
	Inactive	Waiting for Response	Response Sent
Packet arrives request (code 2) New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Start Response Timer 5) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives request (code 7) New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 4) Start Response Timer 5) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	Not Applicable	No Action	1) Update Record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Transition to Waiting for App Response
Packet arrives Existing Transaction ID Existing Sequence #	Not Applicable	1) Notify: Dup. Arrived (Optional) 2) Send ACK_REQ	1) Notify: Dup. Arrived (Optional) 2) Resend Response Message
Response Timer Expires	Not Applicable	1) Notify: Timeout 2) Delete record 3) Transition to Inactive	1) Notify: Timeout 2) Delete record 3) Transition to Inactive
Abort	Error	1) Delete record 2) Transition to Inactive	1) Delete record 2) Transition to Inactive
Send Response	Error	1) Send Response 2) Start Retry Timer 3) Transition to Response Sent	Error
Retry Timeout	Not Applicable	Not Applicable	1) Decrement Retry Count IF Retry Count expired 2) Free Response buffer 3) Stop Response Timer 4) Transition to Inactive ELSE 1) Resend Response message 2) Start Retry Timer
ACK_RESP arrives, sequence number matches stored value	No action	No action	1) Delete record 2) Transition to Inactive
ACK_RESP arrives, sequence number not equal to stored value	No action	No action	No action

The response time is the response time received in the message header.

5.6.1.3.3 Low-end UCMM Server

5.6.1.3.3.1 Functions

The low-end server cannot support the following features:

- perform response message retries
- detect duplicate messages

5.6.1.3.3.2 States

The defined states and their descriptions of the UCMM Low-end Server are listed in Table 293 below :

Table 293: Low-end UCMM Server States

State	Description
Inactive	The record for UCMM transfer does not exist.
Waiting for Response	Data packet arrived with new Transaction ID. New record is created. The Server is waiting for response to be sent by the Server application.

5.6.1.3.3.3 State Transitions

Figure 60 shows the low-end UCMM server state transition diagram.

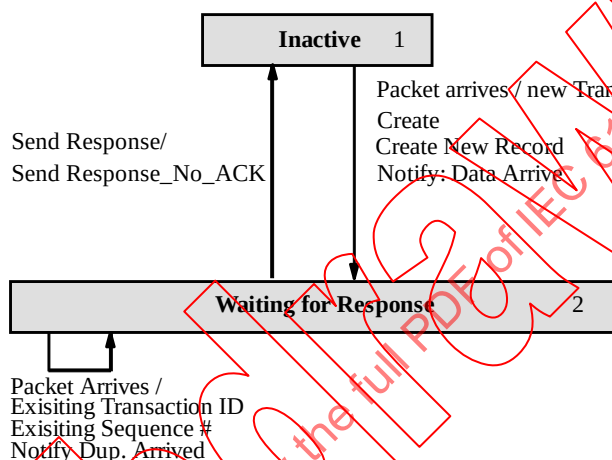


Figure 60: State transition diagram of Low-end UCMM server

5.6.1.3.3.4 State Event Matrix

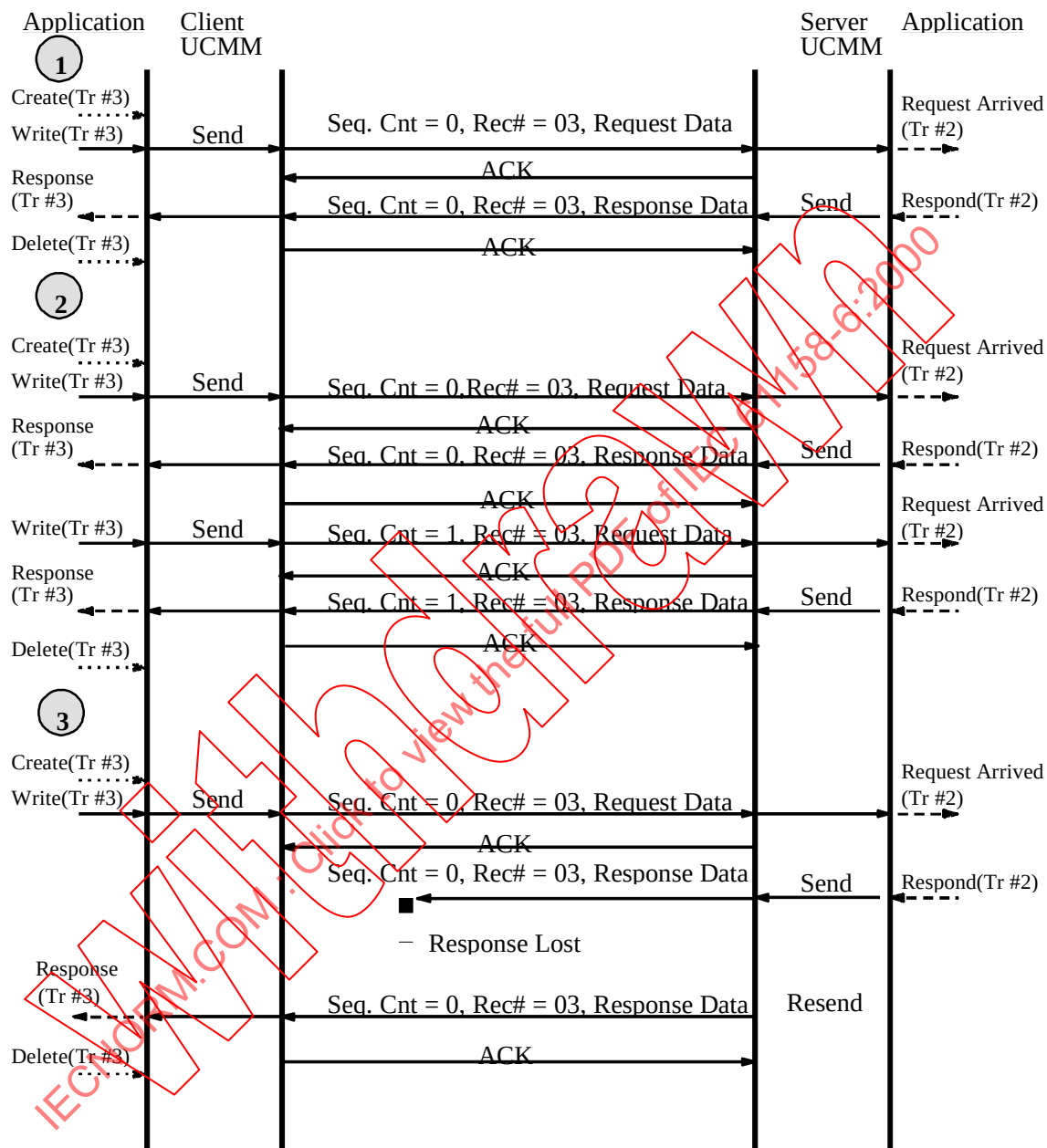
The state transitions for the Low-end UCMM server shall be as specified in in Table 294.

Table 294: State event matrix of Low-end UCMM server

Event	State	
	Inactive	Waiting for Response
Packet arrives New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) IF Response not immediately available, Send ACK_REQ 4) Transition to Waiting for App Response	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	No action	No action
Packet arrives Existing Transaction ID Existing Sequence #	No action	1) Notify: Dup. Arrived (Optional)
Send Response	Error	1) Send Response_No_ACK 2) Transition to Inactive
ACK_RESP arrives	No action	No Action

5.6.1.4 Examples of UCMM Sequences (Informative)

Figure 61 shows a sequence diagram for a UCMM with one outstanding message and Figure 62 shows a sequence diagram for a UCMM with multiple outstanding messages.



Example 1. Typical transaction: Instance is created, request is written, response is returned, and instance is deleted.

Example 2. Two transactions/same instance: In this case the instance is opened and two requests are sent. Note that the client waits for the first response to return before issuing a second request using the same transaction number.

Example 3. Lost response: In this case the response is lost and the server issues a retry.

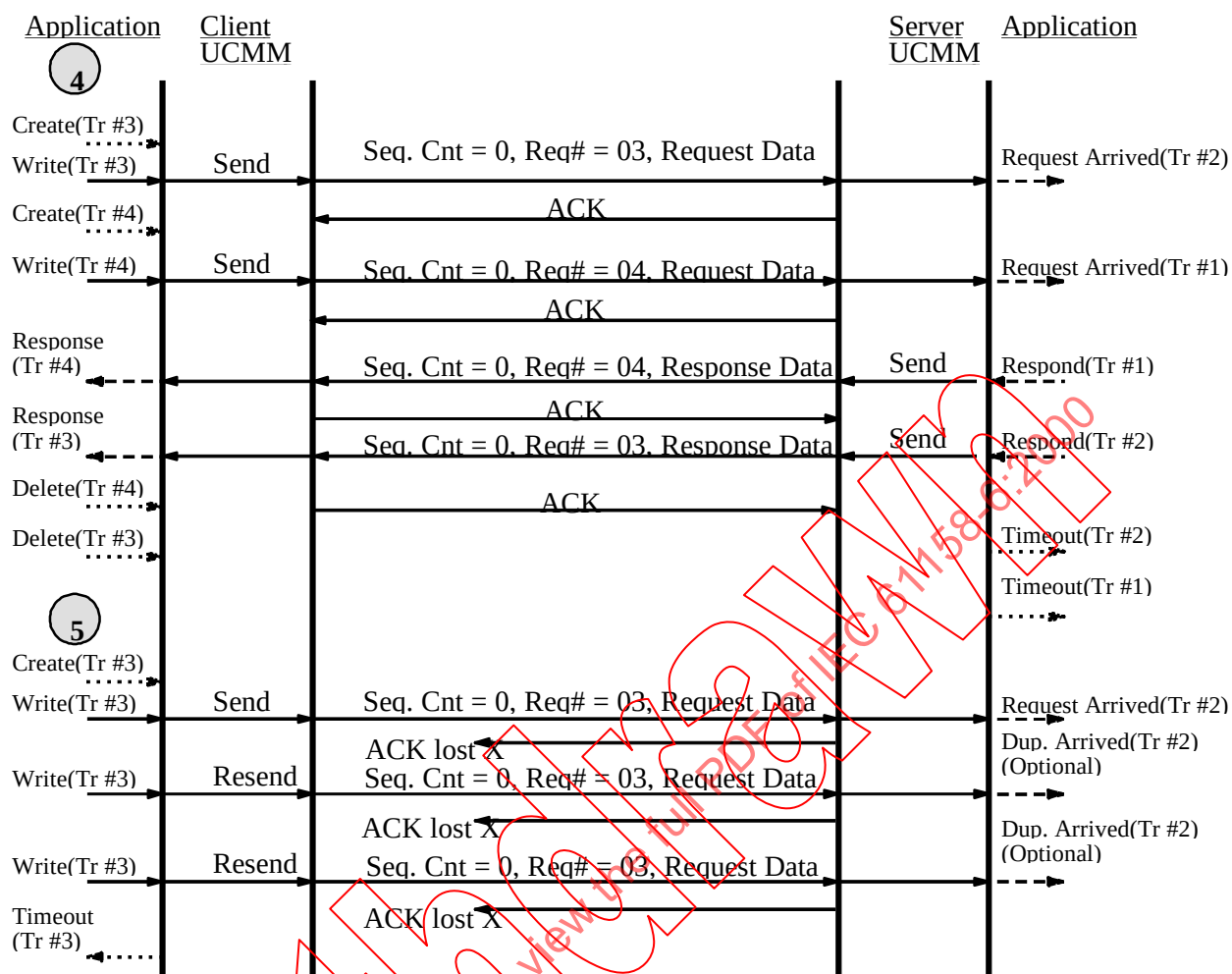


Figure 62: Sequence diagram for a UCMM with multiple outstanding messages

Example 4. Multiple outstanding: In this case the client creates two instances and sends a second request before the first request has completed. Note that the client uses a different transaction number for the second request.

Example 5. Timeout: In this case the server application does not respond and the client issues a timeout. The timeout causes an error status to be returned to the application. Transaction instance shall be deleted.

5.6.1.5 Management UCMM

Any device that has implemented the Keeper object shall also implement the Management UCMM server. With two exceptions, the Management UCMM server shall be identical to the fixed tag 0x83 UCMM server:

- transactions for the Management UCMM server shall be transmitted on fixed tag 0x88;
- the Management UCMM server shall not transmit any packets unless its node contains a Keeper object in the master state.

Any device may implement the Management UCMM client. The Management UCMM client is identical to the fixed tag 0x83 UCMM client except that transactions for the Management UCMM client shall be transmitted on fixed tag 0x88.

NOTE Messages to the Keeper object are broadcast since all devices are permitted to implement this object. To facilitate communication to the Keeper object without impacting non-Keeper devices, these messages are transmitted on a different fixed tag.

5.6.2 Connection-oriented ARPMs (Transports)

5.6.2.1 Transport PDU Buffer

5.6.2.1.1 Transport PDU Format

The T-PDU buffer contains a T-PDU packet. The T-PDU packet consists of a transport header and data. Applications shall write data to and read data from the T-PDU buffers directly. Instances of transports shall write and read the transport header in the T-PDU buffer while consumers shall write data to the T-PDU buffer and producers shall read data from the T-PDU buffer. The T-PDU buffers and buffer management are not part of the communication services. This process is shown in Figure 63.

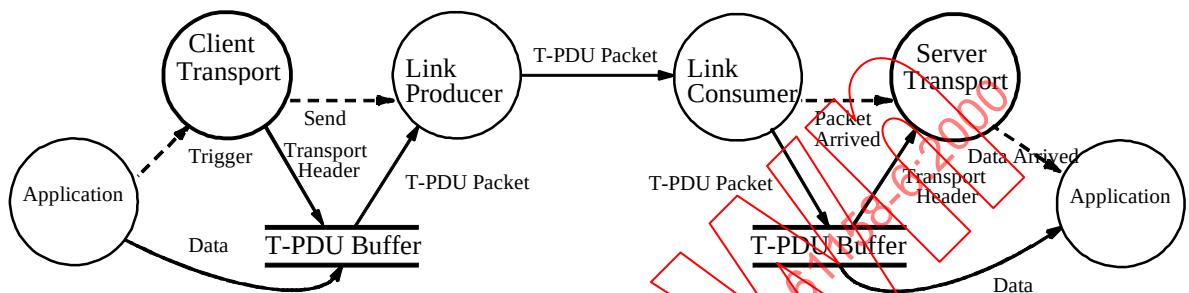


Figure 63: T-PDU Buffer

5.6.2.1.2 Transport PDU Buffer Management

T-PDU buffer management is implementation specific. Implementations that use single buffering, double buffering, overwrite or queuing to manage T-PDU buffers are all allowed.

Implementers are responsible for ensuring buffer integrity (atomic access).

Applications are responsible for ensuring that the buffer has been initialised before a trigger is issued. Failure to initialise the buffer shall allow unknown data to be sent. This is true for producers and the client and server sides.

5.6.2.1.3 Notification

Transport classes can signal the application through events. The valid events are as specified in Table 295 :

Table 295: Notification

Event	Client				Server			
	Class 0	Class 1	Class 2	Class 3	Class 0	Class 1	Class 2	Class 3
Data Arrived					n	n	n	n
Duplicate Arrived						n	n	n
Acknowledgement			n					
Verification				n				

NOTE A *Duplicate Arrived* event is normally not of interest to the application; however, it might be useful in triggering a watchdog timer to indicate that the client's application is alive and triggering.

5.6.2.2 Transport Classes

Applications interface to transport services through the supported transport classes. Table 296 lists the general-purpose transport classes that have been defined for the systems. Each transport class provides a different level of functionality. Transport class 0 provides the minimum functionality required of a transport class, enabling data transfer between applications.

Table 296: Transport Classes

Class Number	Class Name
0	Null (or Base)
1	Duplicate Detection
2	Acknowledged
3	Verified
4	Non-blocking
5	Non-blocking, Fragmenting
6	Multicast, Fragmenting

5.6.2.3 Common Primitive Definitions

Transport classes shall provide the following services :

TR_Write	Writes an item of application data into sending Transport PDU buffer, ready to be sent through the pre-established connection.
TR_Trigger	Causes an item of application data or response data previously placed in sending Transport PDU buffer, to be sent through the pre-established connection.
TR_Packet_arrived	Indicates to the Transport instance that a data packet arrived in the receiving Transport PDU buffer. This service is initiated by local action.
TR_Ack_received	Indicates to the user that a data packet arrived in the receiving Transport PDU buffer indicating an acknowledgement of arrival of previously sent data in the receiving Transport PDU buffer. This service is initiated by local action.
TR_Verify	Indicates to the user that a data packet arrived in the receiving Transport PDU buffer indicating a verification of arrival of previously sent data in the receiving user object. This service is initiated by local action.
TR_Status_Update	Indicates to the user that an updated status is present in the Transport PDU buffer.

Refer to IEC 1158-5 for detailed definition of these services.

Primitives exchanged between Transport instances and FSPM are shown in the following Table 297 and Table 298 .

Table 297: Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
TR_Write_req	FSPM	Transport identifier Application data	Conveys a request from the FSPM to the ARPM to write application data into Transport PDU buffer.
TR_Trigger.req	FSPM	Transport identifier	Conveys a request from the FSPM to the ARPM to trigger transfer of application data from Transport PDU buffer to the Link Producer.
TR_Verify.req	FSPM	Transport identifier	Conveys a request from the FSPM to the ARPM to trigger transfer of write verification data into Transport PDU buffer.

Table 298: Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
TR_Packet_arrived.ind	ARPM	Transport identifier Data arrived Duplicate arrived	Conveys an indication to the FSPM that an item of data has arrived at the Link Consumer.
TR_Ack_arrived.ind	ARPM	Transport identifier	Conveys an indication to the FSPM that an acknowledgement has arrived at the Link Consumer.
TR_Status_Update.ind	ARPM	Transport identifier	Conveys an indication to the FSPM that a new status has arrived at the Link Consumer.
TR_Verify.cnf	ARPM	Transport identifier	Conveys a confirmation from the ARPM to FSPM to indicate reception of verification frame.

5.6.2.4 Parameters of Common Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 299.

Table 299: Parameters used with Primitives Exchanged between FSPM and ARPM

Parameter Name	Description
Transport identifier	Identifier of the instance of the transport invoked in the transaction.
Application data	Contains application data (service request/response parameters or user formatted data).
Data arrived	The Data_Arrived and Duplicate_Arrived parameters indicate whether the new packet contains new data, or whether it is just a duplicate from a previously received packet.

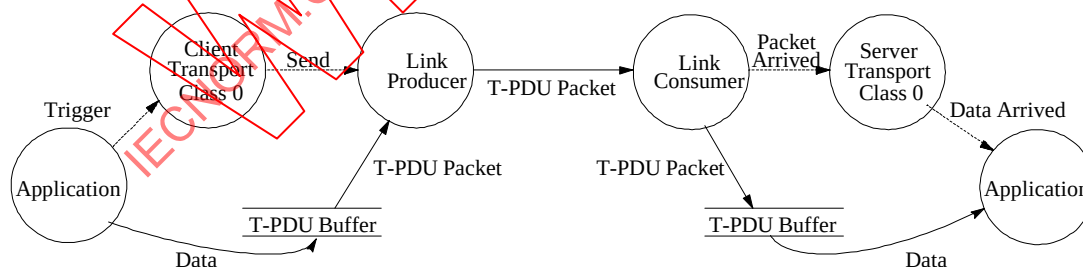
5.6.2.5 Transport State Machines – Class 0

5.6.2.5.1 Functions

Transport class 0 is the simplest transport class, and can use either a point-to-point or multicast connection. Class 0 is typically used to transmit or receive inputs on a multicast connection or outputs on a point-to-point connection.

Possible uses of transport class 0 include sampled inputs, standard outputs, cyclic block transfers, diagnostic events, and communication between controllers and operator interface devices.

Since transport class 0 does not detect duplicate data packets, the target application is responsible for detecting them. A duplicate data packet is a retransmission of the last data packet sent. Figure 64 shows the actions which take place during a data transfer using client transport class 0 and server transport class 0.



NOTE: The client transport instance does not write a sequence count to the client-side T-PDU buffer, and the server transport instance does not read a sequence count from the server-side T-PDU buffer. The transport header is defined as a don't care.

Figure 64: Data Flow Diagram using a client transport class 0 and server transport class 0

Figure 65 shows the sequence in which actions take place when transferring data using client transport class 0 and server transport class 0.

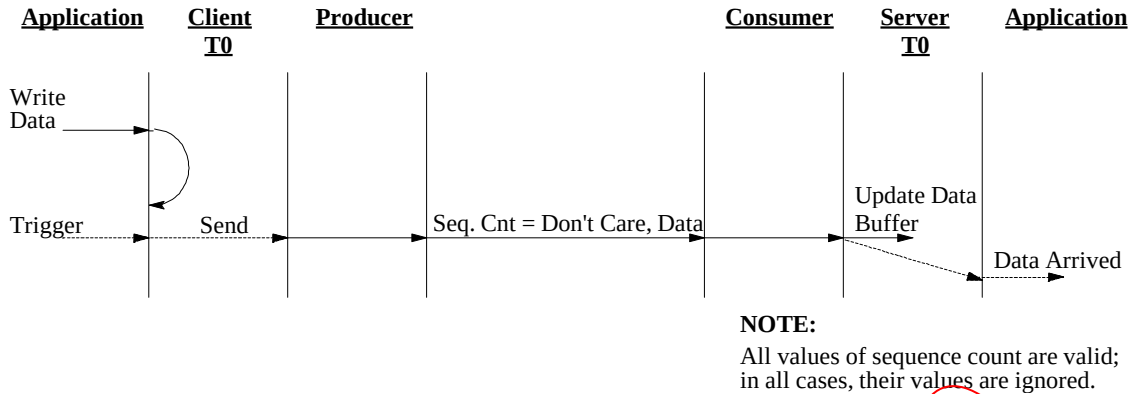


Figure 65: Sequence Diagram of Data Transfer Using Transport Class 0

5.6.2.5.2 Transport Class 0 Client

5.6.2.5.2.1 Functions

The function of client transport class 0 is just to pass the *trigger* service from the application to the producer as a *send* service. It is useful for the client transport class 0 to be shown from a high-level design view. In actual implementations, the application may trigger the producer directly. An idle state is provided for consistency with other transport classes. In the *idle* state all events, except *delete* and *start*, are ignored.

A class 0 client transport is responsible for:

- Accepting the client application's trigger that it has written a data packet to the client-side T-PDU buffer
- Writing a transport header to the T-PDU buffer for each packet that the client application writes to the client-side T-PDU buffer (optional)
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection

5.6.2.5.2.2 States

The defined states and their descriptions of the Class 0 Transport Client are listed in Table 300 below :

Table 300: Class 0 Transport Client States

State	Description
Non-existent	The instance of Transport Class 0 does not exist.
Idle	The instance of Transport Class 0 has been created but not yet started.
Running	The instance of Transport Class 0 has been created and started.

5.6.2.5.2.3 State Transitions

State transition diagram of the class 0 client transport is shown in Figure 66.

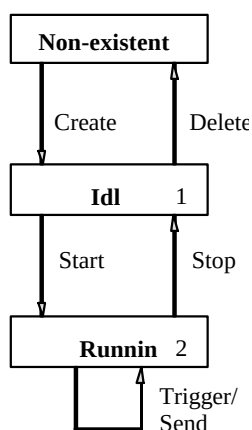


Figure 66: Class 0 client STD

5.6.2.5.2.4 State Event Matrix

State event matrix of the class 0 client transport is shown in Table 301.

Table 301: Class 0 client SEM

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Running	Error
Stop	Error	No Action	Transition to Idle
Trigger	Error	No Action	Send

5.6.2.5.3 Transport Class 0 Server

5.6.2.5.3.1 Function

The function of server transport class 0 is to pass the *packet arrived* event from the consumer to the producer as a *data arrived* event. It is useful for the server transport class 0 to be shown from a high-level design view. In actual implementations, the application may receive the event directly from the consumer. In the *idle* state all packet arrivals are ignored.

A class 0 server transport is responsible for:

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU buffer
- Locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side T-PDU buffer
- Notifying the server application that data has been written to the server-side T-PDU buffer

5.6.2.5.3.2 States

The defined states and their descriptions of the Class 0 Transport Server are listed in Table 302 below:

Table 302: Class 0 Transport Server States

State	Description
Non-existent	The instance of Transport Class 0 does not exist.
Idle	The instance of Transport Class 0 has been created but not yet started.
Running	The instance of Transport Class 0 has been created and started.

5.6.2.5.3.3 State Transitions

State transitions of the class 0 server transport are shown in Figure 67

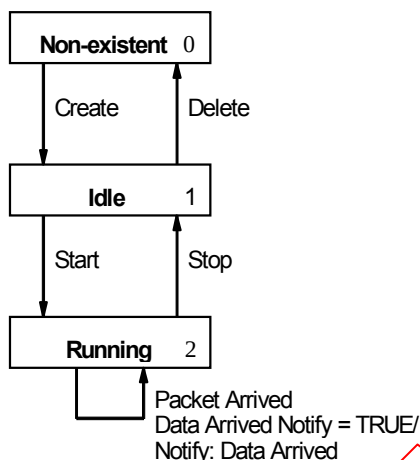


Figure 67: Class 0 server STD

5.6.2.5.3.4 State Event Matrix

State event matrix of the class 0 server transport is shown in Table 303.

Table 303 Class 0 server SEM

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Running	Error
Stop	Error	No action	Transition to Idle
Packet Arrived & Data Arrived Notify = TRUE	No action	No action	Notify: Data Arrived

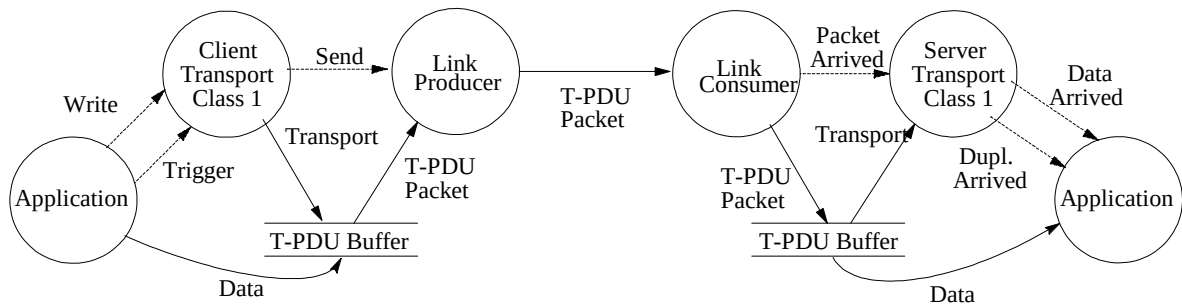
5.6.2.6 Transport State Machines – Class 1

5.6.2.6.1 Functions

Transport class 1, like class 0, shall allow either a point-to-point or multicast connection. Unlike class 0, which has no transport header, the class 1 transport shall include a sequence number, which is used to detect the delivery of duplicate data packets.

NOTE Class 1 is preferred to class 0 for targets because the target application does not have to perform duplicate detection, which reduces the overhead required in those end nodes that support the change of state transport trigger. Class 1 permits an efficient change of state trigger, since was designed to allow change of state data transfers.

Figure 68 shows the actions which take place during data transfer using a class 1 client transport instance and a class 1 server transport instance.

**NOTE:**

The T-PDU packet comprises the transport header from the client transport and data from the application.

Figure 68: Data Flow Diagram Using Client Transport Class 1 and Server Transport Class 1

Figure 69 shows the sequence in which actions take place when transferring data using client transport class 1 and server transport class 1. Note that the sequence count is incremented with every write. Also note that in the case where the packet is lost on a new data sample, and the packet is later triggered and sent, the data received by the client is treated as new data. This is because this is the first time the server has received this sequence count. This mechanism provides fault tolerance for those samples that change infrequently.

NOTE It is an implementation choice to decide whether to update the data buffer upon receipt of duplicate data or not, based on the potential impact of the additional processing.

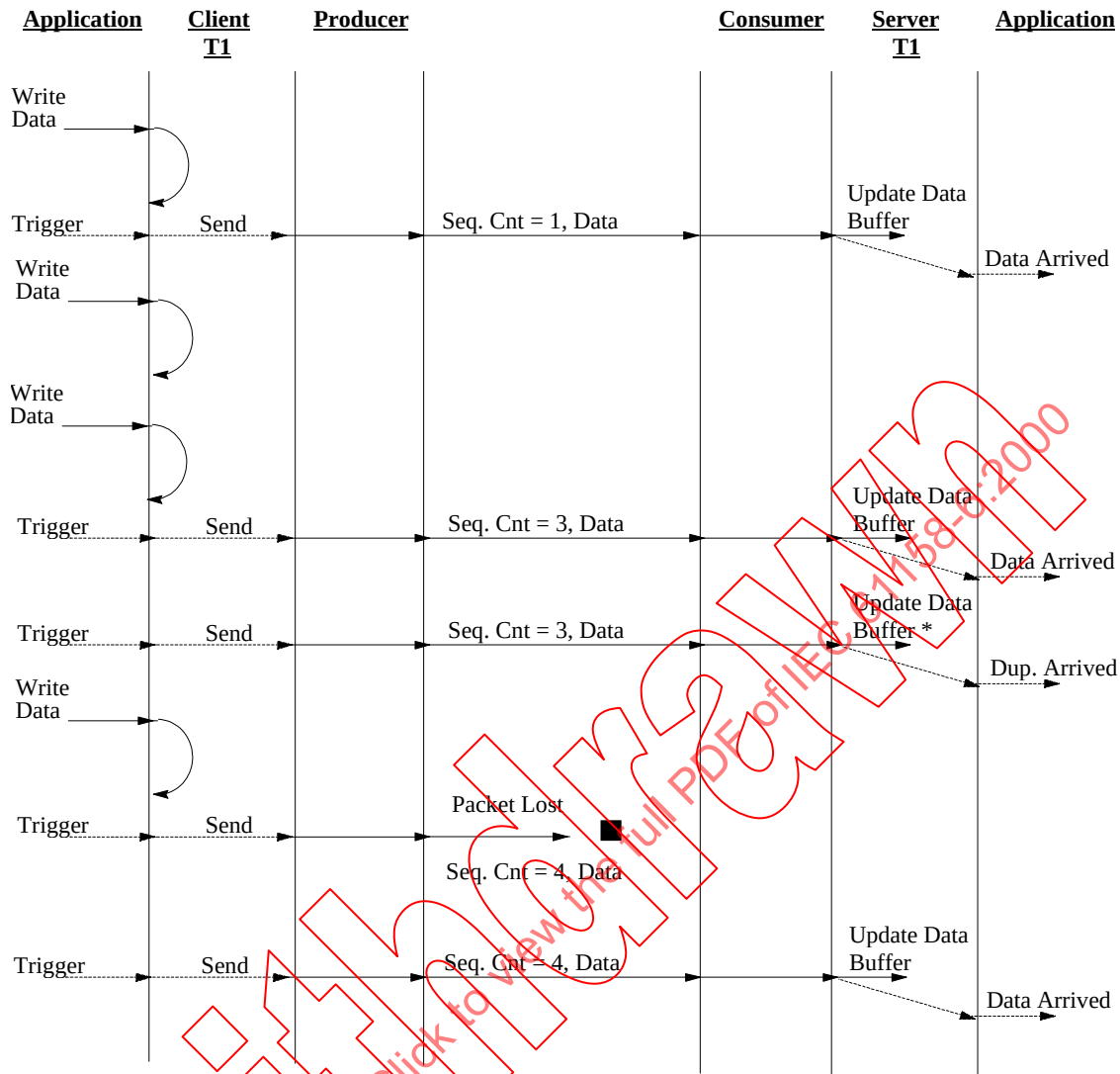


**Figure 69: Sequence Diagram of Data Transfer
Using Client Transport Class 1 and Server Transport Class 1**

Transport class 1 can be used in conjunction with a heartbeat timer to keep a connection open when there are periods when no data is transmitted. The heartbeat timer shall be initialised every time the producing node of the network connection transmits a packet, and shall be set so that its maximum value, or the maximum time between data transmissions, is less than the path time-out value for the network connection. When the heartbeat timer reaches its maximum value, it triggers the client transport instance so that the producing node produces and retransmits the packet in the client-side T-PDU buffer before the connection times out. The consuming node of the network connection, recognising that the packet has the same sequence count as the last packet it received, may or may not overwrite the packet in the server-side T-PDU buffer; that implementation decision is left to the product developers, since they can best assess the impact of additional processing.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, a timer may be used to trigger the production of the current data in the T-PDU buffer. To transmit new data the application would first write to the T-PDU buffer and then trigger the client transport.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, the data arrived and duplicated arrived events may be ORed together to reset a timer. If this timer were to expire, the application would know that no data was received within the designated time. The data arrived event would identify new

samples of data. This may allow applications to reduce their overhead by only processing new data samples.

Possible uses of transport class 1 include sampled inputs, standard outputs, cyclic block transfers, diagnostic events, and communication between controllers and operator interface devices.

5.6.2.6.2 Transport Class 1 Client

5.6.2.6.2.1 Functions

Transport class 1 starts with the behaviour in class 0 and adds a sequence count. This sequence count is incremented by the client transport class when data is written to the data buffer. When the transport receives a *write* event, it shall increment the sequence count. When a *trigger* event is received the transport shall update the sequence count in the T-PDU buffer and invoke the *send* service.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, a timer may be used to trigger the production of the current data in the T-PDU buffer. To transmit new data the application would first write to the T-PDU buffer and then trigger the client transport.

A class 1 client transport is responsible for:

- Accepting the client application's trigger that it has written a data packet to the client-side T-PDU buffer
- Writing a transport header to the T-PDU buffer for each packet that the client application writes to the client-side T-PDU buffer.
- Initialising the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission)
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection

5.6.2.6.2.2 States

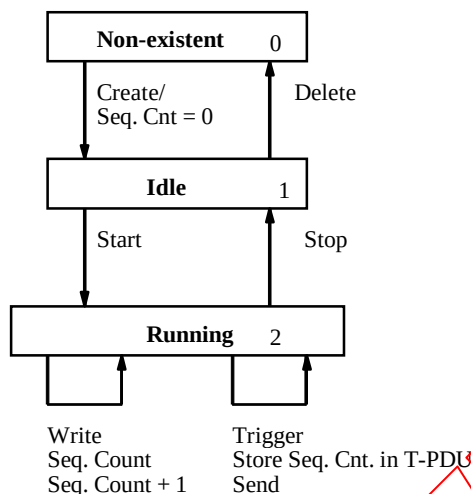
The defined states and their descriptions of the Class 1 Transport Client are listed in Table 304 below :

Table 304: Class 1 Transport Client States

State	Description
Non-existent	The instance of Transport Class 1 does not exist.
Idle	The instance of Transport Class 1 has been created but not yet started.
Running	The instance of Transport Class 1 has been created and started.

5.6.2.6.2.3 State Transitions

State transitions of the class 1 client transport are shown in Figure 70.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every write event until it reaches a maximum value of $2^{16}-1$. After that it rolls over to 0.

Figure 70: Class 1 client STD

5.6.2.6.2.4 State Event Matrix

State transitions of the class 1 client transport are shown in Table 305.

Table 305: Class 1 client SEM

Event	State		
	Non-existent	Idle	Running
Create	1) Transition to Idle 2) Set seq. count = 0	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Running	Error
Stop	Error	No Action	Transition to Idle
Write	Error	No Action	1) Seq. Cnt = Seq. Cnt + 1
Trigger	Error	No Action	1) Store Seq. Cnt. in T-PDU Buffer 2) Send

5.6.2.6.3 Transport Class 1 Server

5.6.2.6.3.1 Functions

Transport class 1 starts with the behaviour in class 0 and adds a sequence count. The received sequence count is compared with the previous sequence count. If they are equal, the received packet is considered to be a duplicate packet. If they are not equal, the received data is considered to be a new sample. There is no attempt to count how many sequence counts have changed between samples.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, the data arrived and duplicated arrived events may be ORed together to reset a timer. If this timer were to expire, the application would know that no data was received within the designated time. The data arrived event would identify new samples of data. This may allow applications to reduce their overhead by only processing new data samples.

A class 1 server transport is responsible for

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU buffer;
- Locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side T-PDU buffer;
- Triggering the server application that data has been written to the server-side T-PDU buffer;
- Comparing the sequence count of each packet with that of the previous packet;
- Notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match.

5.6.2.6.3.2 States

The defined states and their descriptions of the Class 1 Transport Server are listed in Table 306 below :

Table 306: Class 1 Transport Server States

State	Description
Non-existent	The instance of Transport Class 1 does not exist.
Idle	The instance of Transport Class 1 has been created but not yet started.
Ready to Run	The instance of Transport Class 1 has been created and started, waiting for a first data packet to arrive.
Running	The instance of Transport Class 1 has been created and started, first data packet has arrived.

5.6.2.6.3.3 State Transitions

State transitions of the class 1 server transport are shown in Figure 71.

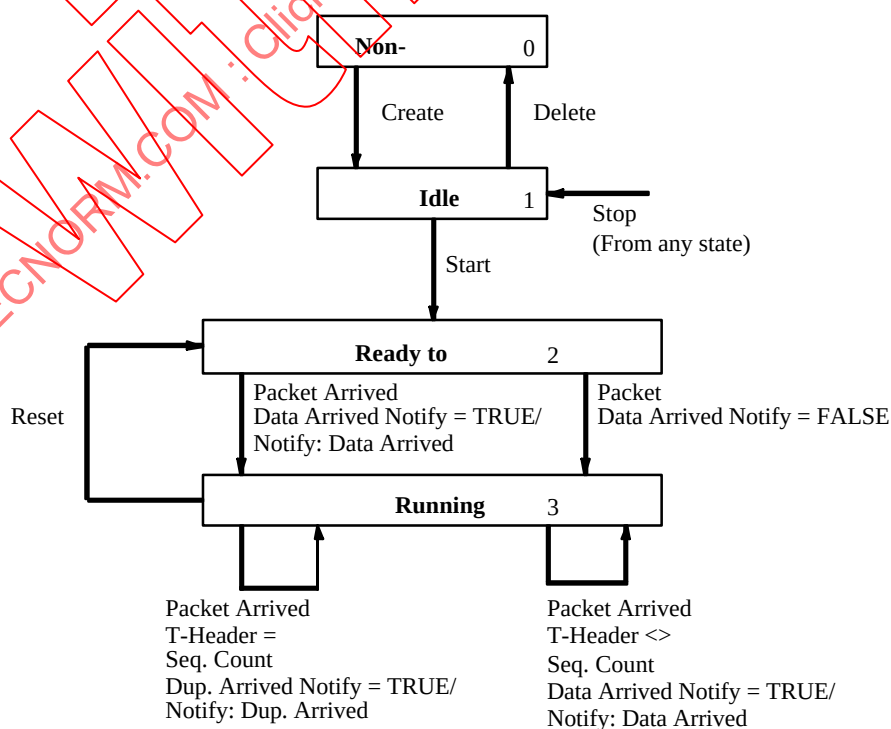


Figure 71: Class 1 server STD

5.6.2.6.3.4 State Event Matrix

State transitions of the class 1 server transport are shown in Table 307.

Table 307: Class 1 server SEM

Event	State/			
	Non-existent	Idle	Ready to Run	Running
Create	Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Ready to Run	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle
Reset	Error	Error	No Action	Transition to Ready to Run
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running Notify: Data Arrived	Notify: Data Arrived
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running Notify: Data Arrived	Notify: Duplicate Arrived
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running Notify: Data Arrived	Notify: Data Arrived
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running Notify: Data Arrived	No action
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running	No action
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running	Notify: Duplicate Arrived
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running	No action
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running	No action

5.6.2.7 Transport State Machines – Class 2

5.6.2.7.1 Functions

Transport class 2 starts with the behaviour in class 1 and adds a return connection that acknowledges the delivery of a packet. The producer connection from the client node can be point to point or multicast. The connection from the server to the client is point to point.

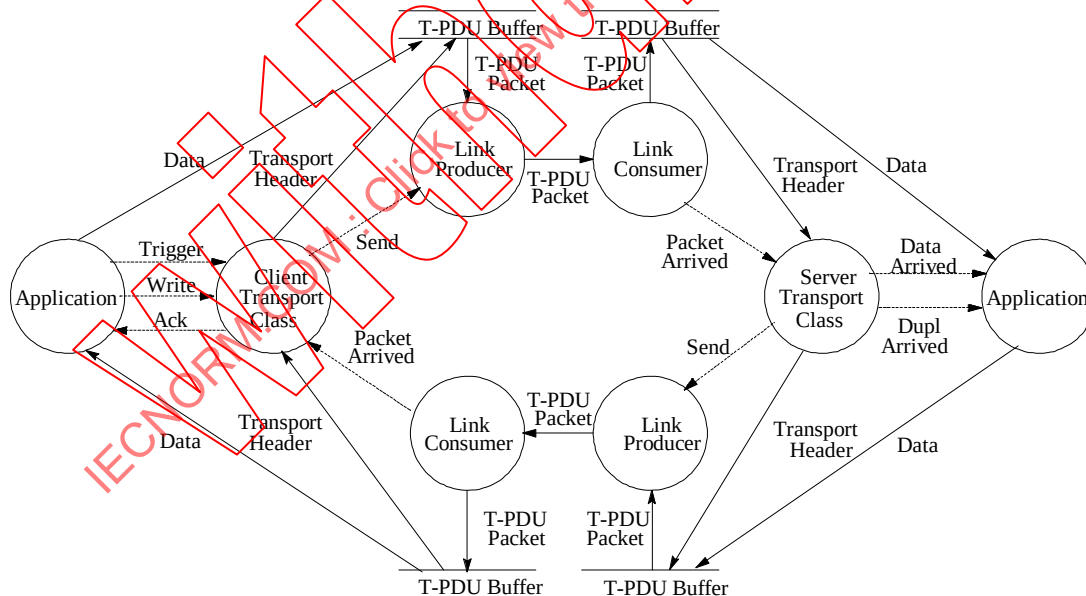
This transport class allows the client application to be notified that the server has received the transmitted packet. This acknowledgement tells the client application only that the data arrived. This does not mean that the server application has read the buffer or that a new sample can be sent without overwriting the buffer.

The main advantage of this class over class 3 (Verified) is that the acknowledgement is returned immediately after the packet arrives at the consumer, while the verify class requires the application to initiate the return verification. The delays associated with receiving the acknowledgement are small, approximately twice the time that is required for a one way transmission.

Transport class 2 uses two network connections: the originator-to-target connection, which can be either point-to-point or multicast; and the return connection, which shall be point-to-point. A multicast connection using transport class 2 shall have a point-to-point return connection corresponding to each client-to-server connection. **The acknowledgement notifies the client transport instance that the consuming node of the network connection has received the packet.**

If a server transport instance does not acknowledge receipt of the packet, the client transport instance triggers the client application to retransmit the most recently sent packet; if the client-to-server connection is one of the network connections of a multicast, data is resent over all of the network connections of the multicast. The server application can return data to the client application along with its acknowledgement.

Figure 72 shows the actions which take place during a data transfer using client transport class 2 and server transport class 2



NOTES:

- 1 The client and server transports each use two buffers: one for transmitting, and one for receiving.
- 2 The target application is not involved in sending the acknowledgement.

Figure 72: Data Flow Diagram Using Client Transport Class 2 and Server Transport Class 2

The server transport instance initiates and sends the acknowledgement to the client transport instance to notify the client application that it can write the next data packet to the client-side T-PDU transmit buffer. Each acknowledgement shall be received by the client transport instance before the next packet is sent over the same client-to-server network connection(s).

Figure 73 shows the sequence of actions which take place during data transfer using client transport class 2 and server transport class 2. In the example depicted, the acknowledgement is the only data transmitted in the server-to-client direction. The unshaded areas in Figure 73 indicate client-to-server data transmission, and the shaded areas indicate server-to-client transmission.

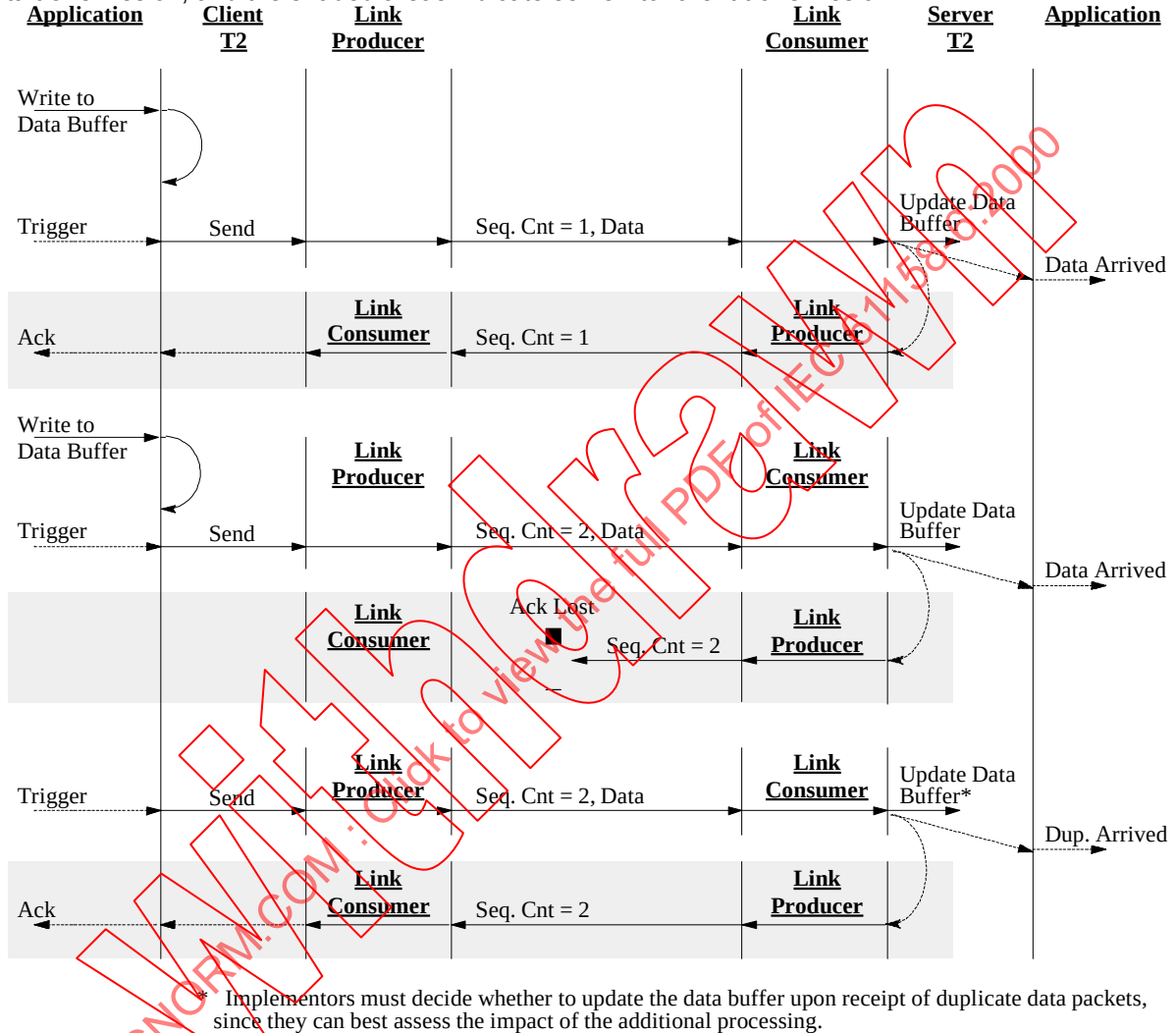


Figure 73: Diagram of Data Transfer Using Client Transport Class 2 and Server Transport Class 2 without Returned Data

At times, the server-to-client transmission comprises the acknowledgement *and* additional data.

Figure 74 shows the sequence in which actions take place during data transfer using client transport class 2 and server transport class 2 when the acknowledgement and data are returned to the client. Note that the returned data is written asynchronously to the receipt of data from the client. The unshaded areas in Figure 74 indicate client-to-server data transmission, and the shaded areas indicate server-to-client data transmission.

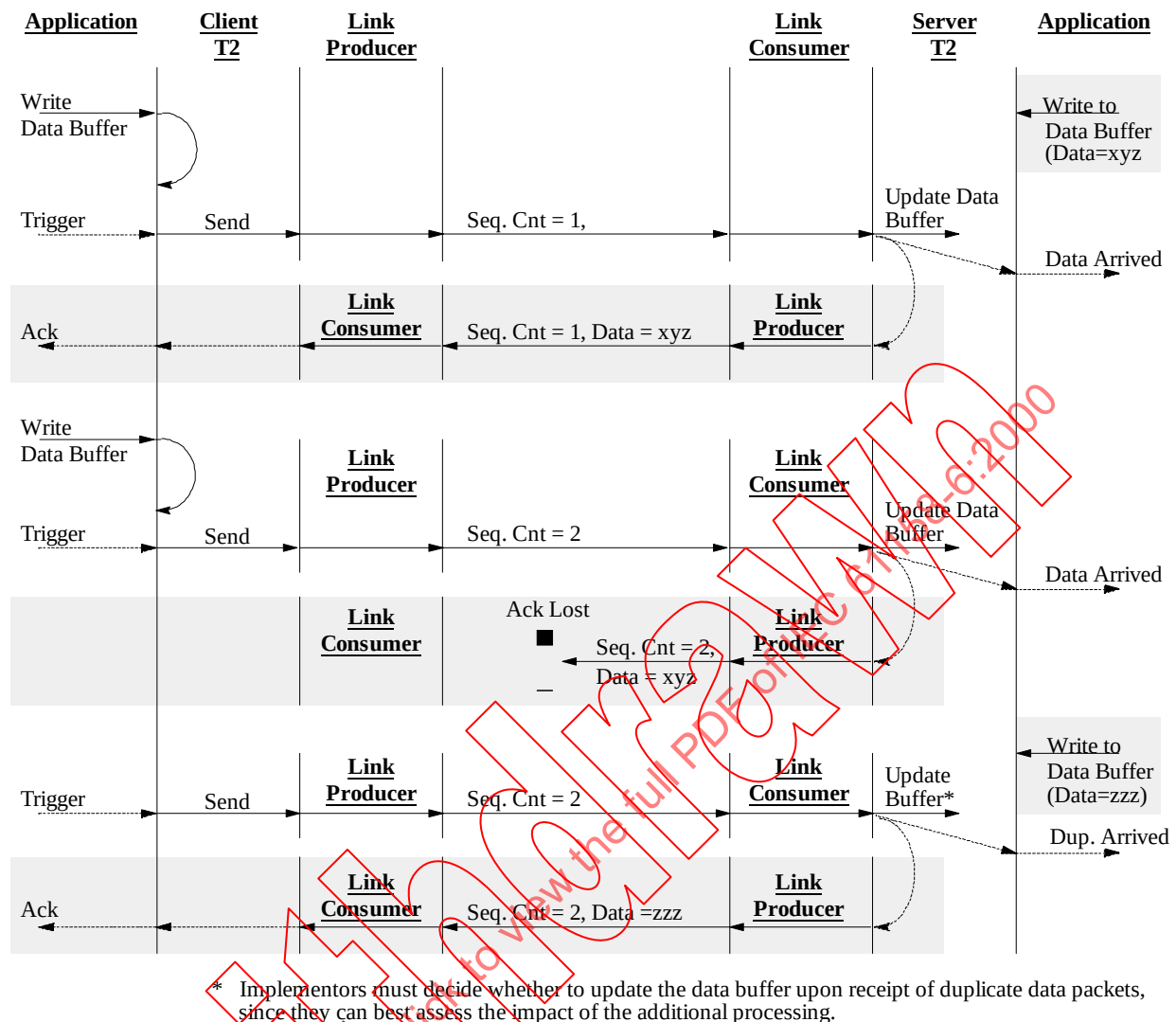


Figure 74: Sequence Diagram of Data Transfer Using Client Transport Class 2 and Server Transport Class 2 with Returned Data

Possible uses of transport class 2 include master/slave applications and communication between controllers and operator interface devices.

5.6.2.7.2 Transport Class 2 Client

5.6.2.7.2.1 Functions

In the client-to-server direction, a class 2 client transport is responsible for:

- Accepting the client application's trigger that it has written a data packet to the client-side T-PDU transmit buffer;
- Writing a transport header to the client-side T-PDU transmit buffer for each packet that the client application writes to it (NOTE: The transport header for each packet shall include a sequence count.);
- Initialising the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection.

In the server-to-client direction, a class 2 client transport is responsible for:

- Accepting notification from the consuming node of the return network connection that it has written data (the server transport instance's acknowledgement) to the client-side T-PDU receive buffer;
- Locating and reading the transport header of each data packet that the consuming node of the return network connection writes to the client-side T-PDU receive buffer;
- Notifying the client application that the consuming node of the return network connection has written data to the client-side T-PDU receive buffer;

Note that the application can update the data buffer or retrigger data before all of the acknowledgements have been received. If the client transport is retriggered while waiting for an acknowledgement, the previous acknowledgement received register is cleared. This means that the client transport class shall wait for all active consumers to return an acknowledgement before notifying the application.

5.6.2.7.2.2 States

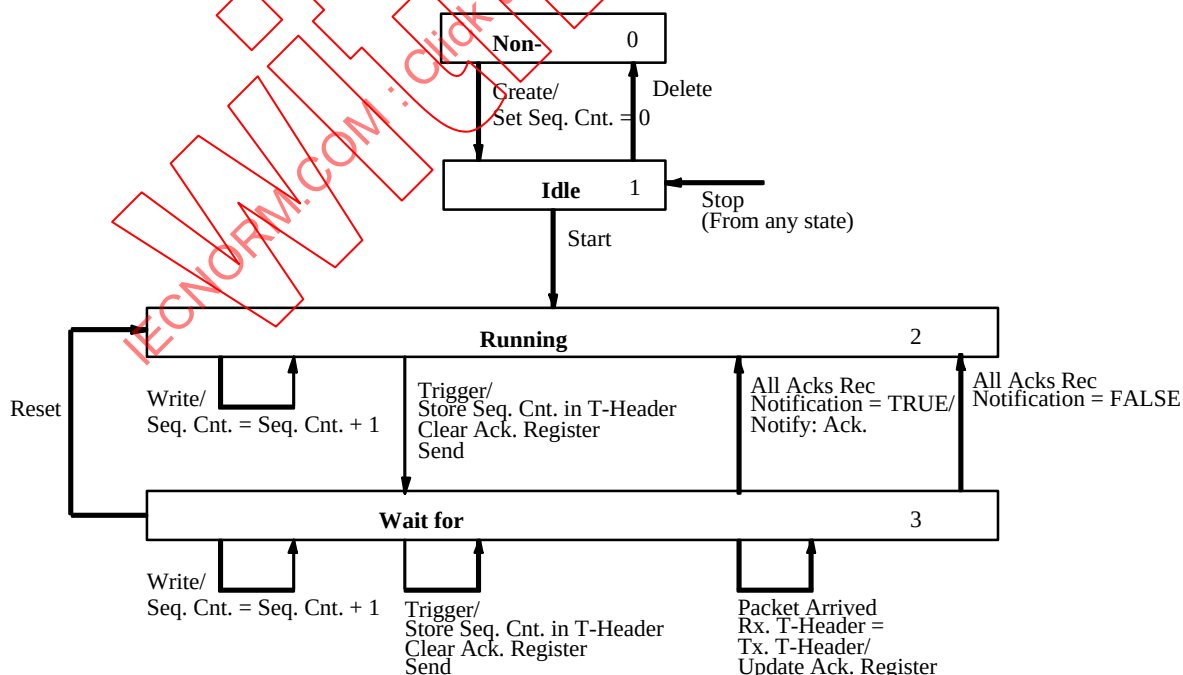
The defined states and their descriptions of the Class 2 Transport Client are listed in Table 308 below :

Table 308: Class 2 Transport Client States

State	Description
Non-existent	The instance of Transport Class 2 does not exist.
Idle	The instance of Transport Class 2 has been created but not yet started.
Running	The instance of Transport Class 2 has been created and started.
Wait for Acknowledgement	The instance of Transport Class 2 has been created and started, acknowledgement is pending.

5.6.2.7.2.3 State Transitions

State transitions of the class 2 client transport are shown in Figure 75.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of $2^{16}-1$. After that it rolls over to 0.

Figure 75: Class 2 client STD

5.6.2.7.2.4 State Event Matrix

State transitions of the class 2 client transport are shown in Table 309.

Table 309: Class 2 client SEM

Event	State			
	Non-existent	Idle	Running	Wait for ack
Create	1) Seq. Cnt. = 0 2) Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Running	Error	Error
Stop	Error	No Action	Transition to Idle	Transition to Idle
Reset	Error	Error	No Action	Transition to Running
Write	Error	No Action	1) Seq. Cnt. = Seq. Cnt. + 1 2) write T-PDU	1) Seq. Cnt. = Seq. Cnt. + 1 2) write T-PDU
Trigger	Error	No Action	1) Store Seq. Cnt. in T-PDU 2) Clear Ack. register 3) Send 4) Transition to Wait for Ack.	1) Store Seq. Cnt. in T-PDU 2) Clear Ack. Register 3) Send
Packet Arrived Rx. T-Header = Tx. T-Header	Error	No Action	No Action	1) Update Ack. Register
All Ack Received Ack. Notify = TRUE	Error	No Action	Error	1) Notify: Ack. 2) Transition to Running
All Ack Received Ack. Notify = FALSE	Error	No Action	Error	1) Transition to Running

NOTE The STD and SEM for client classes 2 and 3 are nearly identical. Verify has been substituted for acknowledgement.

5.6.2.7.3 Transport Class 2 Server

5.6.2.7.3.1 Functions

Server transport class 2 starts with the behaviour from transport class 1 and adds a return connection from the server to the client that is used to acknowledge a message. This return connection is used to identify the acknowledgement message.

After receiving a *packet arrived* event, the server transport shall read the received transport header, store it in the transmit transport header and invoke the *send* service. The application is not involved in the acknowledgement. The producer connection from the client node can be point to point or multicast. The connection from the server to the client is point to point.

In the client-to-server direction, a class 2 server transport is responsible for

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU receive buffer;
- Locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side T-PDU receive buffer;
- Comparing the sequence count of each packet in the server-side T-PDU receive buffer with that of the previous packet;
- Notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match;
- Notifying the server application that data has been written to the server-side T-PDU receive buffer. (ACK may contain server data)

In the server-to-client direction, a class 2 server transport is responsible for

- Writing the transport header (including sequence count) of the data in the server-side T-PDU receive buffer to the server-side T-PDU transmit buffer;
- Triggering the producing node of the return network connection to produce the T-PDU packet on the link and send it to the consuming node of the return network connection.

5.6.2.7.3.2 States

The defined states and their descriptions of the Class 2 Transport Server are listed in Table 310 below :

Table 310: Class 2 Transport Server States

State	Description
Non-existent	The instance of Transport Class 2 does not exist.
Idle	The instance of Transport Class 2 has been created but not yet started.
Running	The instance of Transport Class 2 has been created and started.

5.6.2.7.3.3 State Transitions

State transitions of the class 2 server transport are shown in Figure 76

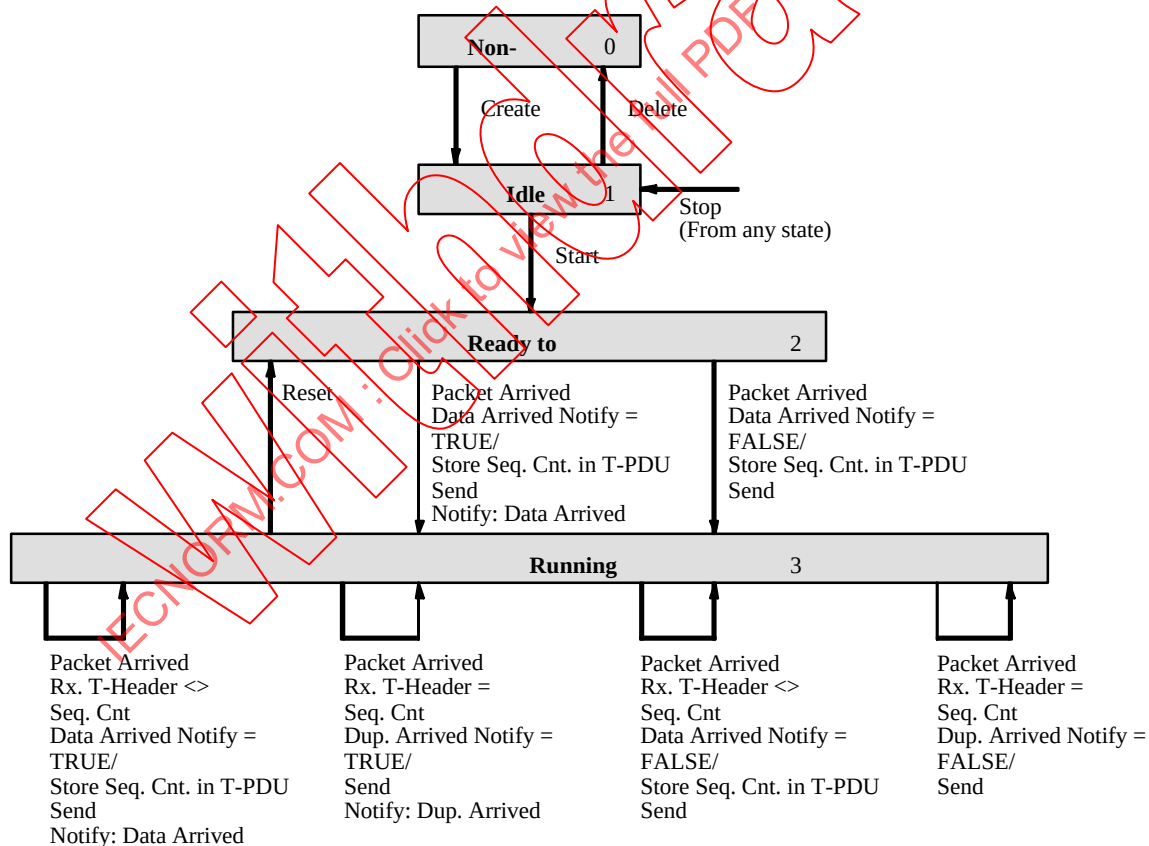


Figure 76: Class 2 server STD

NOTE The send service is invoked for sequence counts that match and sequence counts that differ. This behaviour is required to ensure that a node that has sent an acknowledgement that was lost shall retransmit the acknowledgement on a retry.

5.6.2.7.3.4 State Event Matrix

State transitions of the class 2 server transport are shown in Table 311.

Table 311: Class 2 server SEM

Event	State			
	Non-existent	Idle	Ready to Run	Running
Create	Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Ready to Run	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle
Reset	Error	Error	No action	Transition to Ready to Run
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Notify: Data Arrived
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Send (Implies Ack) 2) Notify: Duplicate Arrived
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Notify: Data Arrived
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Send (Implies Ack)
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Transition to Running	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack)
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Transition to Running	1) Send (Implies Ack) 2) Notify: Duplicate Arrived
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Transition to Running	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack)
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. T-PDU 2) Send (Implies Ack) 3) Transition to Running	1) Send (Implies Ack)

5.6.2.8 Transport State Machines – Class 3

5.6.2.8.1 Functions

Transport class 3 starts with the behaviour in class 1 and adds a return connection that verifies that the application has received the packet. The sequence count that shall be returned with the verify is the same as the sequence count sent with the data. This return connection is used to identify the verification message.

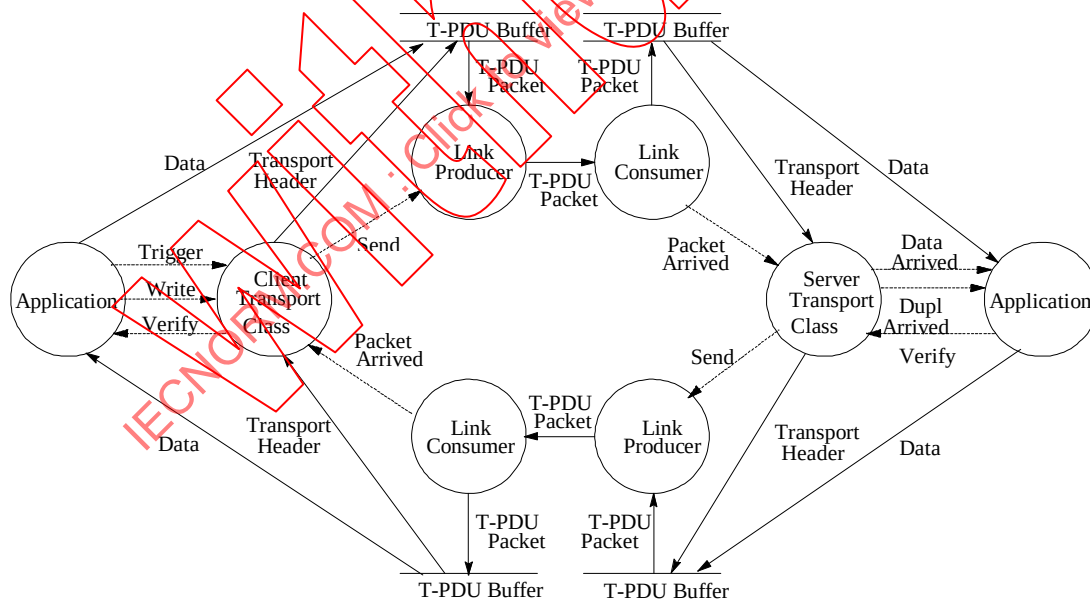
This transport class allows the application to be notified that the server application has responded to the transmitted packet. This verification tells the client application not only that the data arrived but that the server application has read the buffer and that a new sample can be sent without overwriting the buffer.

The main advantage of this class over class 2 (Acknowledgement) is that verification conveys an application response. The disadvantage is that the delay in receiving the verification can be very difficult to predict.

Transport class 3 uses two connections: the originator-to-target connection, which can be either point-to-point or multicast; and the return connection, which shall be point-to-point. A multicast connection using transport class 3 shall have a point-to-point return connection corresponding to each client-to-server connection. **The verification notifies the client application that the server application has received and read the transmitted data.** If a server application does not verify receipt of the packet, the client application retransmits the most recently sent packet; if the client-to-server connection is a multicast, the data is retransmitted over all of the network connections of the multicast.

The server application initiates and sends the verification after the consuming node of the network connection has written the packet to the T-PDU buffer, after the transport has read the packet's transport header, and after the application has read the packet. The server's verification signals the client application that the server-side T-PDU receive buffer is ready to accept the next data packet.

Figure 77 shows the actions which take place during a data transfer using client transport class 3 and server transport class 3.

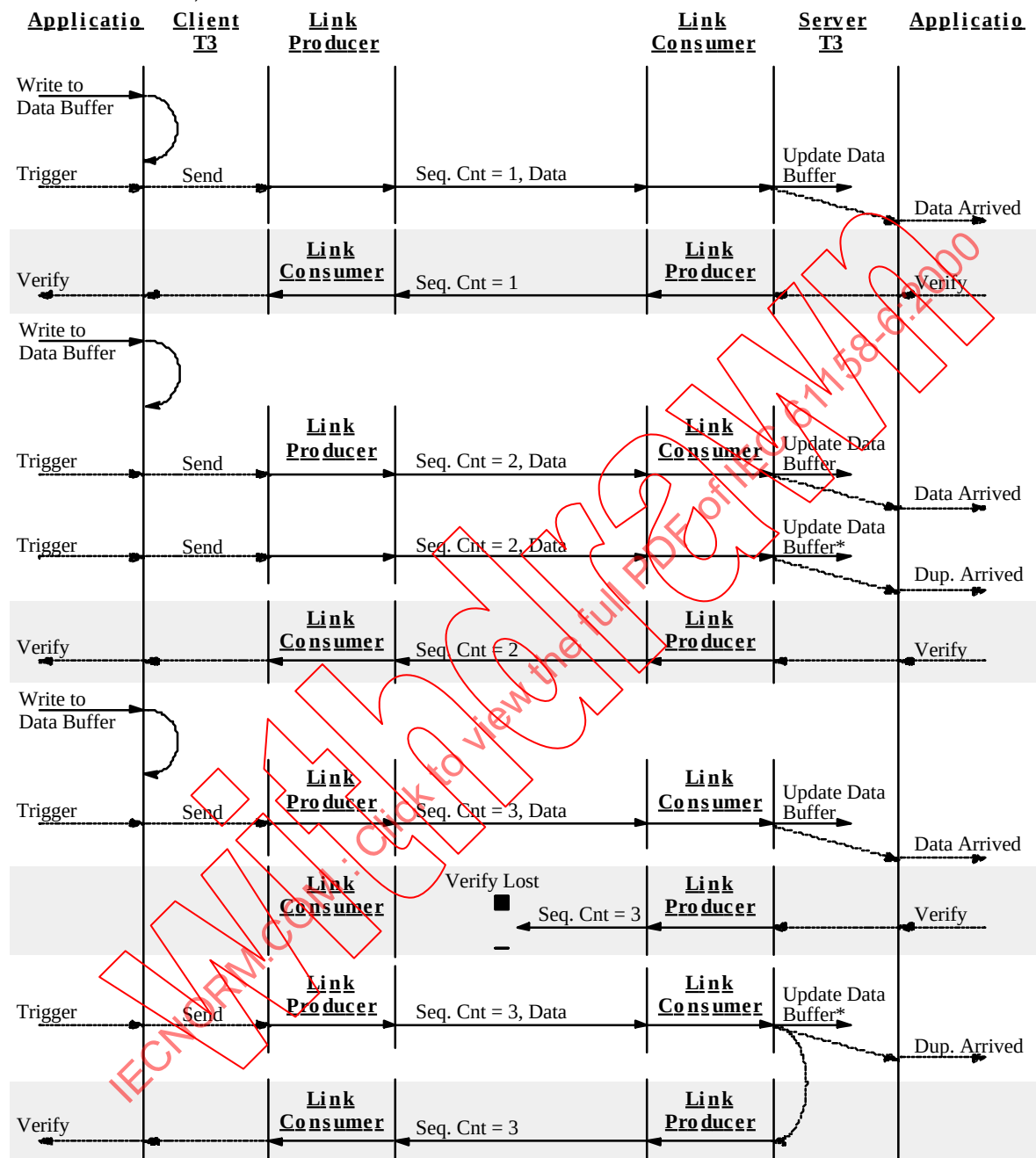


NOTES.

- 1 The client and server transports each use two buffers: one for transmitting, and one for receiving.
- 2 The server application can return data to the client application using the connection established for the verification.

Figure 77: Data Flow Diagram Using Client Transport Class 3 and Server Transport Class 3

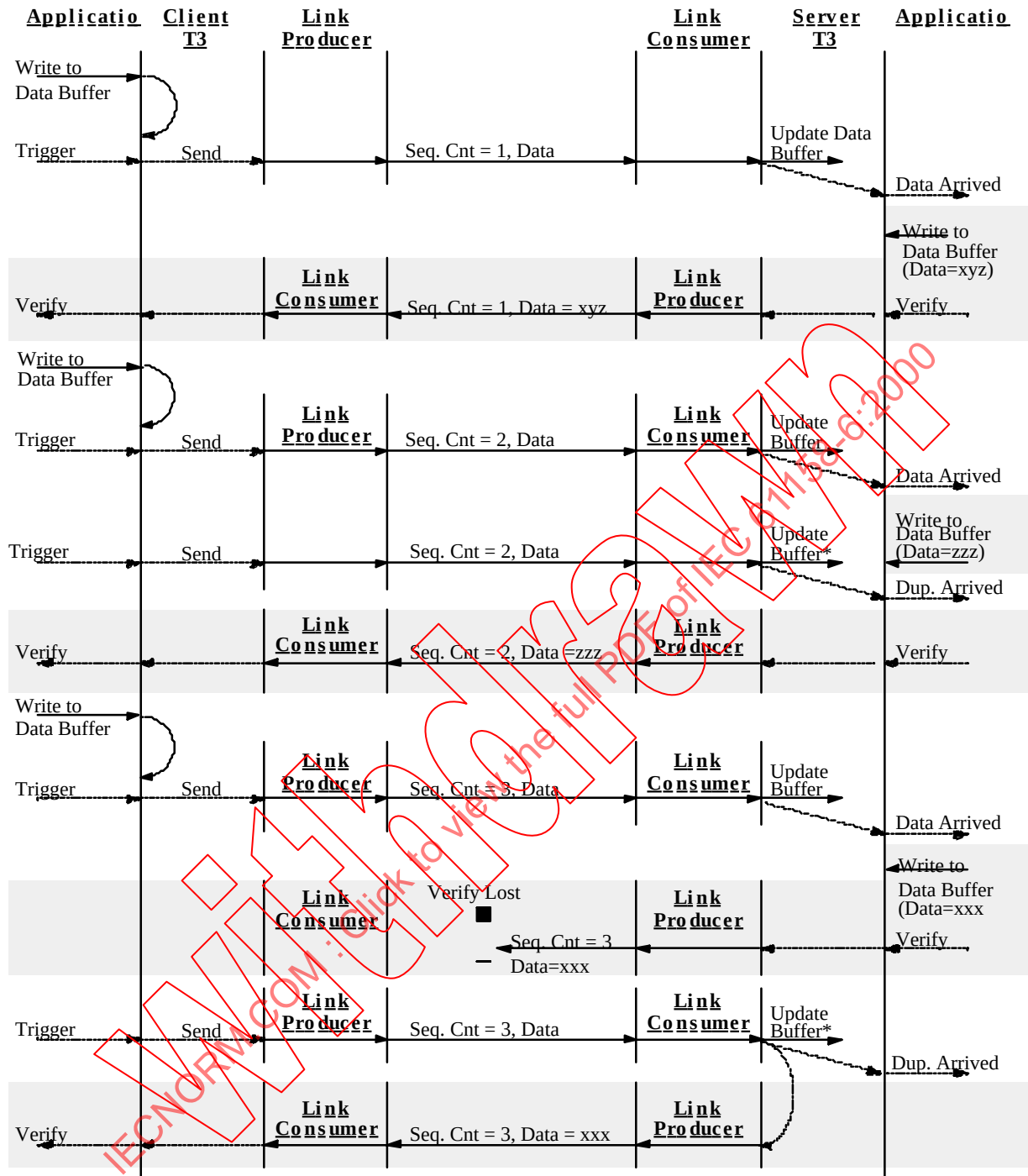
Figure 78 shows the sequence of actions which take place during data transfer using client transport class 3 and server transport class 3. In the example depicted, the verification is the only data transmitted in the server-to-client direction. The unshaded areas in Figure 78 indicate client-to-server data transmission, and the shaded areas indicate server-to-client transmission.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data packets, since they can best assess the impact of the additional

Figure 78: Sequence diagram of data transfer using client transport class 3 and server transport class 3 without returned data

At times, the server-to-client transmission comprises the verification *and* additional data. Figure 79 shows the sequence in which actions take place during data transfer using client transport class 3 and server transport class 3 when the verification and data are returned to the client. Note that the returned data is written asynchronously to the receipt of data from the client. The unshaded areas in Figure 79 indicate client-to-server data transmission, and the shaded areas indicate server-to-client transmission.



* Implementors must decide whether to update the data buffer upon receipt of duplicate data since they can best assess the impact of the additional processing.

Figure 79: Sequence diagram of data transfer using client transport class 3 and server transport class 3 with returned data

Possible uses of transport class 3 include change-of-state inputs and outputs, asynchronous [independent] block transfers, event acknowledgements, communication between controllers and operator interface devices, uploads, downloads, and messaging.

5.6.2.8.2 Transport Class 3 Client

5.6.2.8.2.1 Functions

In the client-to-server direction, a class 3 client transport is responsible for:

- Accepting the client application's trigger that it has written a data packet to the client-side T-PDU transmit buffer;
- Writing a transport header to the client-side T-PDU transmit buffer for each packet that the client application writes to it;
- Initialising the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection.

In the server-to-client direction, a class 3 client transport is responsible for:

- Accepting notification from the consuming node of the return network connection that it has written data (the server transport instance's verification) to the client-side T-PDU receive buffer;
- Locating and reading the transport header of each data packet that the consuming node of the return network connection writes to the client-side T-PDU receive buffer;
- Notifying the client application that the consuming node of the return network connection has written data to the client-side T-PDU receive buffer.

Note that the application can update the data buffer or retrigger data before all of the verifications have been received. If the client transport is retriggered while waiting for verification, the previous verification received register is cleared. This means that the client transport class shall wait for all active consumers to return verification before notifying the application.

5.6.2.8.2.2 States

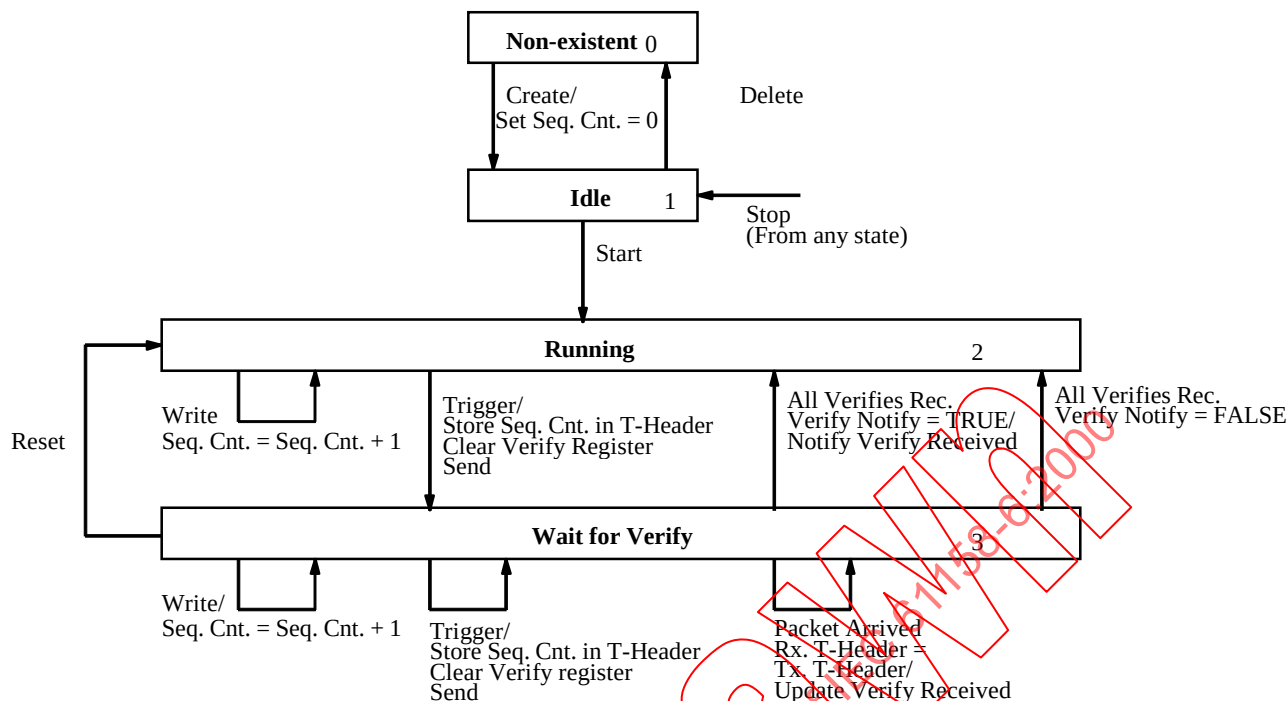
The defined states and their descriptions of the Class 3 Transport Client are listed in Table 312 below :

Table 312: Class 3 Transport Client States

State	Description
Non-existent	The instance of Transport Class 3 does not exist.
Idle	The instance of Transport Class 3 has been created but not yet started.
Running	The instance of Transport Class 3 has been created and started.
Wait to verify	The instance of Transport Class 3 has been created and started, verification is pending.

5.6.2.8.2.3 State Transitions

State transitions of the class 3 client transport are shown in Figure 80.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of $2^{16}-1$. After that it rolls over to 0.

Figure 80: Class 3 client STD

5.6.2.8.2.4 State Event Matrix

State transitions of the class 3 client transport are shown in Table 313.

Table 313: Class 3 client SEM

Event	State			
	Non-existent	Idle	Running	Wait for verify
Create	1) Seq. Cnt. = 0 2) Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Running	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle
Reset	Error	Error	No Action	Transition to Running
Write	Error	No action	Seq. Cnt. = Seq. Cnt. + 1	Seq. Cnt. = Seq. Cnt. + 1
Trigger	Error	No action	1) Store Seq. Cnt. in T-PDU 2) Clear Verify register 3) Send 4) Transition to Wait for Verify	1) Store Seq. Cnt. in T-PDU 2) Clear Verify register 3) Send
Packet Arrived Rx. T- header = Tx. T-Header	No action	No action	No Action	1) Update Verify Register
All Verifies Received Notification = TRUE	No action	No action	Error	1) Notify: Verify 2) Transition to Running
All Verifies Received Notification = FALSE	No action	No action	Error	1) Transition to Running

NOTE The STD and SEM for client classes 2 and 3 are nearly identical. Acknowledgement has been substituted for verification.

5.6.2.8.3 Transport Class 3 Server

5.6.2.8.3.1 Functions

Transport class 3 starts with the behaviour in class 1 and adds a return connection that verifies that the application has received the packet. It is the responsibility of the application to verify receipt of data. The sequence count that shall be returned with the verification is the same as the sequence count sent with the data.

The application is required to read the T-PDU buffer before invoking the verify service.

In the client-to-server direction, a class 3 server transport is responsible for:

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU receive buffer;
- Locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side T-PDU receive buffer;
- Comparing the sequence count of each packet in the server-side T-PDU receive buffer with that of the previous packet;
- Notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match;
- Notifying the server application that data has been written to the server-side T-PDU receive buffer.

In the server-to-client direction, a class 3 server transport is responsible for:

- Writing the transport header of the data in the server-side T-PDU receive buffer to the server-side T-PDU transmit buffer;
- Correlating that transport header with any data that the server application wants to send to the client application and has written to the server-side T-PDU transmit buffer;
- Triggering the producing node of the return network connection to produce the T-PDU packet on the link and send it to the consuming node of the return network connection;

NOTE Data from the application can be return from the server to the client along the verification connection.

The send service is invoked when verify is received from an application or when a duplicate transmission is received while in the running state. Retransmitting when a previously verified packet has been received allows this class to recover from lost verifications.

5.6.2.8.3.2 States

The defined states and their descriptions of the Class 3 Transport Server are listed in Table 314 below :

Table 314: Class 3 Transport Server States

State	Description
Non-existent	The instance of Transport Class 3 does not exist.
Idle	The instance of Transport Class 3 has been created but not yet started.
Read to run	The instance of Transport Class 3 has been created and started, waiting for the first packet to arrive.
Wait for verify	The instance of Transport Class 3 has been created and started, the first packet arrived, waiting for verification to be requested by the application.
Running	The instance of Transport Class 3 has been created and started, verification has been processed.

5.6.2.8.3.3 State Transitions

State transitions of the class 3 server transport are shown in Figure 81.

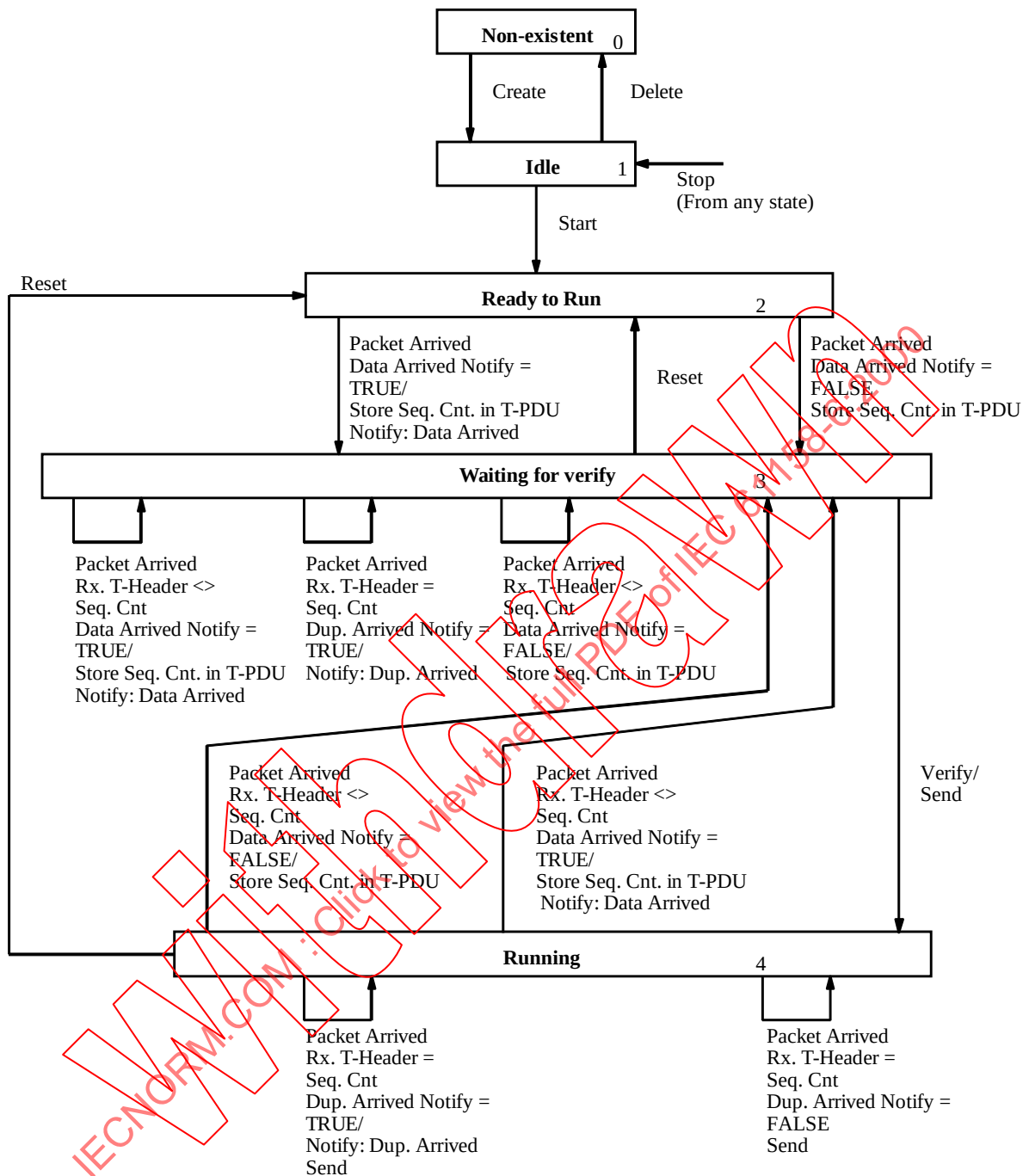


Figure 81: Class 3 server STD

5.6.2.8.3.4 State Event Matrix

State transitions of the class 3 server transport are shown in Table 315.

Table 315: Class 3 server SEM

Event	State				
	Non-existent	Idle	Ready to Run	Waiting for verify	Running
Create	Transition to Idle	Error	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error	Error
Start	Error	Transition to Ready to Run	Error	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle	Transition to Idle
Reset	Error	Error	No action	Transition to Ready to Run	Transition to Ready to Run
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = TRUE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Notify: Dup. Arrived	1) Notify: Dup. Arrived 2) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = TRUE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	No action	1) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = FALSE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Transition to Waiting for Verify	1) Store Seq. Cnt. in T-PDU	1) Store Seq. Cnt. in T-PDU 2) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Transition to Waiting for Verify	1) Notify: Dup. Arrived	1) Notify: Dup. Arrived 2) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq. Count Data Arrived Notify = FALSE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Transition to Waiting for Verify	1) Store Seq. Cnt. in T-PDU	1) Store Seq. Cnt. in T-PDU 2) Transition to Waiting for Verify

Event	State				
	Non-existent	Idle	Ready to Run	Waiting for verify	Running
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in T-PDU 2) Transition to Waiting for Verify	No action	1) Send (Implies Verify)
Verify	Error	No action	No action	1) Send (Implies Verify) 2) Transition to Running	No action

5.6.2.9 Transport State Machines – Classes 4, 5, 6

This subclause explains the common elements of the transport classes 4 – 6, which provide higher-level features such as non-blocking, fragmentation/assembly, and transport-level acknowledgement. Transport classes 4 and 5 shall not be used with messaging. Messaging does not provide a way to match requests with responses. Use class 3 for messaging without fragmentation.

The fragmentation protocols assume that per-connection send order is preserved throughout the system; overwrite is permissible, but changing the order of the messages is not.

5.6.2.9.1 Class 4 and 5 Data Flow Diagram

Figure 82 shows the relationship of the class 4 and 5 Transport to the Application and the Link Producer and Link Consumer, the data that move between these functions, and the events which are generated and used by them. (The class 6 data flow diagram is included with the class 6 definition, see 5.6.2.12.2.) Note some significant differences between these and the class 3 transport.

- The application data is entered into a queue, not placed directly into the T-PDU Buffer. This allows the application to give several requests to the transport without waiting for the acknowledgement, verification, or response to a previous request.
- The success or failure of an application request is placed into a queue for the application to access, rather than using a simple event. This is done because the execution of the application and the delivery of the application data are asynchronous.
- The events and data on the right hand side and the left hand side are identical. There is not a “Client” and “Server” transport for these classes. Both sides are capable of sending and receiving application data.
- The trigger timer is explicitly identified, rather than being a portion of the application. This is because the transport is using the trigger signal to perform timing functions.

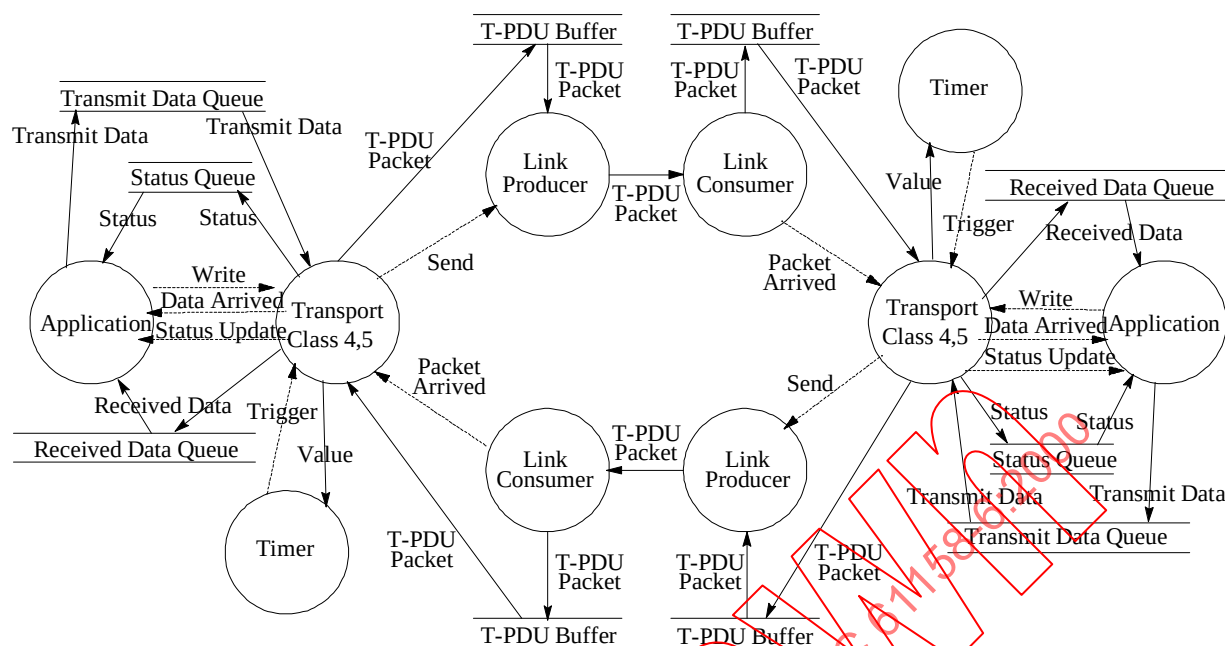


Figure 82: Data Flow Diagram Using Transport Classes 4 and 5

The Write event is used to notify the transport that new application data has been placed in the queue. The Trigger event is assumed to be provided by a resettable retry timer. These events are used in a different sense than they are used for classes 0 to 3. These are shown in Table 316.

Table 316: Write and Trigger Events in Class 4 and 5 Transport

	Classes 0 to 3	Classes 4 to 6
Write event	"New data available": Causes sequence count to increment.	"New data available": Indicates queue is non-empty. If the transport is not otherwise busy, shall cause data to be sent.
Trigger event	Causes data to be sent.	Causes retry or keep alive to be sent.

5.6.2.9.2 Classes 4 and 5 Sequence Diagrams (Informative)

5.6.2.9.2.1 Typical Sequences (Informative)

Some typical sequences are illustrated to aid understanding of the operation of these transport classes. Formal specifications of these transport classes are given in IEC 61158-5.

5.6.2.9.2.2 Typical Message Exchange Sequence (Informative)

Figure 83 shows the sequence in which actions take place during a messaging request/response exchange using transport classes 4 and 5. The unshaded areas in Figure 83 indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction. Each transmission shows the Command and Sequence Number used for each header (SND and RCV). SND gives the command in the direction of the data flow, and RCV is the acknowledgement of the command in the other direction.

When a transport has some data to send, it uses the *Only* command and when it has nothing to send, it uses the *Null* command. It is recommended that the Sender state machine send zero application data bytes with the *Null* command. It is also recommended that the Receiver state machine accept *Null* commands with a non-zero number of application data bytes, which it ignores.



It is permissible for a transport to send two data packets in a row. This implies that the second one might overwrite the first one in an intermediate module. This is desirable, since the second packet contains information that is newer than the first one and the data in the first one is no longer needed.

Figure 84 also shows a different situation where an overwrite can occur. The right side sends a *Null* #2 with the ACK #2 as the result of the keep-alive *Trigger* event. But, before it can be delivered, the right side receives the *Only* #3 from the left side. The left side immediately sends the *Null* #2, but this time with the newer ACK #3, which overwrites the ACK #2.

It is not necessary to wait for the ACK to a *Null* before sending an *Only*.

When sending an *Only*, it shall have a different sequence number than the *Null*, so sender knows which is being ACK'd by the other side.

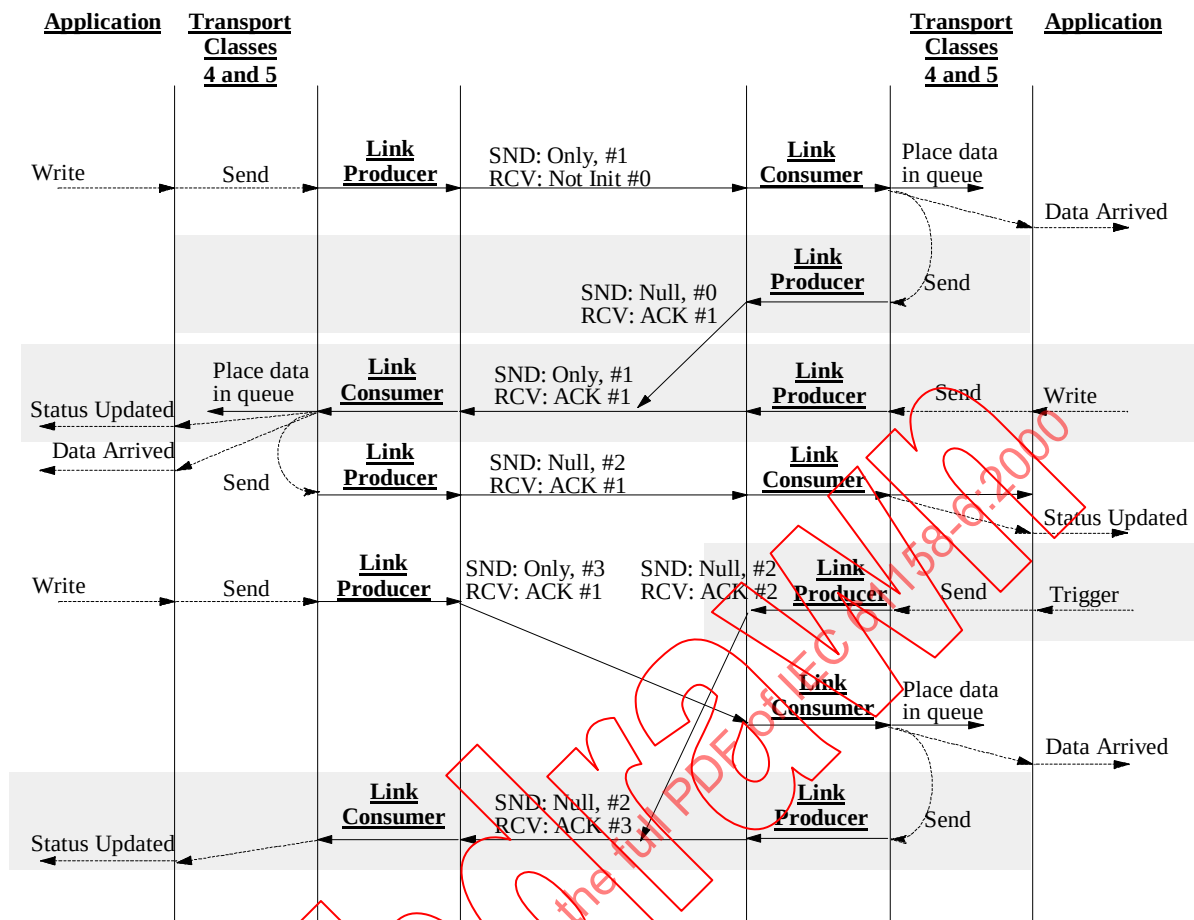


Figure 84: Sequence Diagram of Messages overwriting each other

5.6.2.9.2.4 Typical Queued Sequence (Informative)

Figure 85 shows the sequence in which actions take place during a messaging request/response exchange using transport classes 4 and 5. In this case, the applications place some data into the send queue before the first data transfer has been acknowledged. The unshaded areas in Figure 85 indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

In this case, the left transport shall not use the *Null* #2 to deliver the ACK #1, but shall send the data from its queue via the *Only* #2.

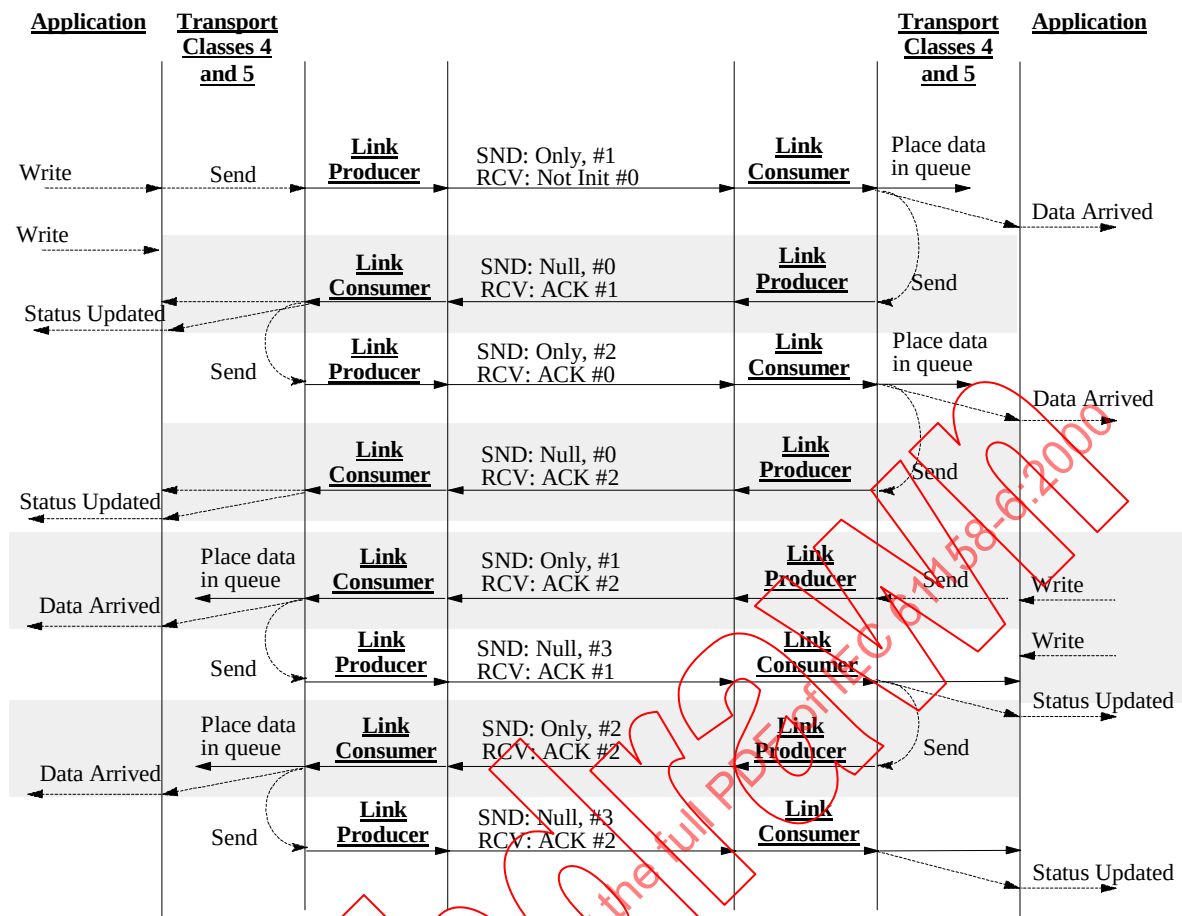


Figure 85: Sequence Diagram of Queued Message Exchange Using Transport Classes 4 and 5

5.6.2.9.2.5 Typical Sequence With Retries (Informative)

Retries are performed when a transport does not receive a Receiver header with the same sequence count as its current Sender header in the timeout period. The T-PDU is not changed for a retry. Figure 86 shows two retry situations: where an *Only* command fails to be delivered and where a *Null* with an expected ACK fails to be delivered. The Trigger event is generated by the retry timer.

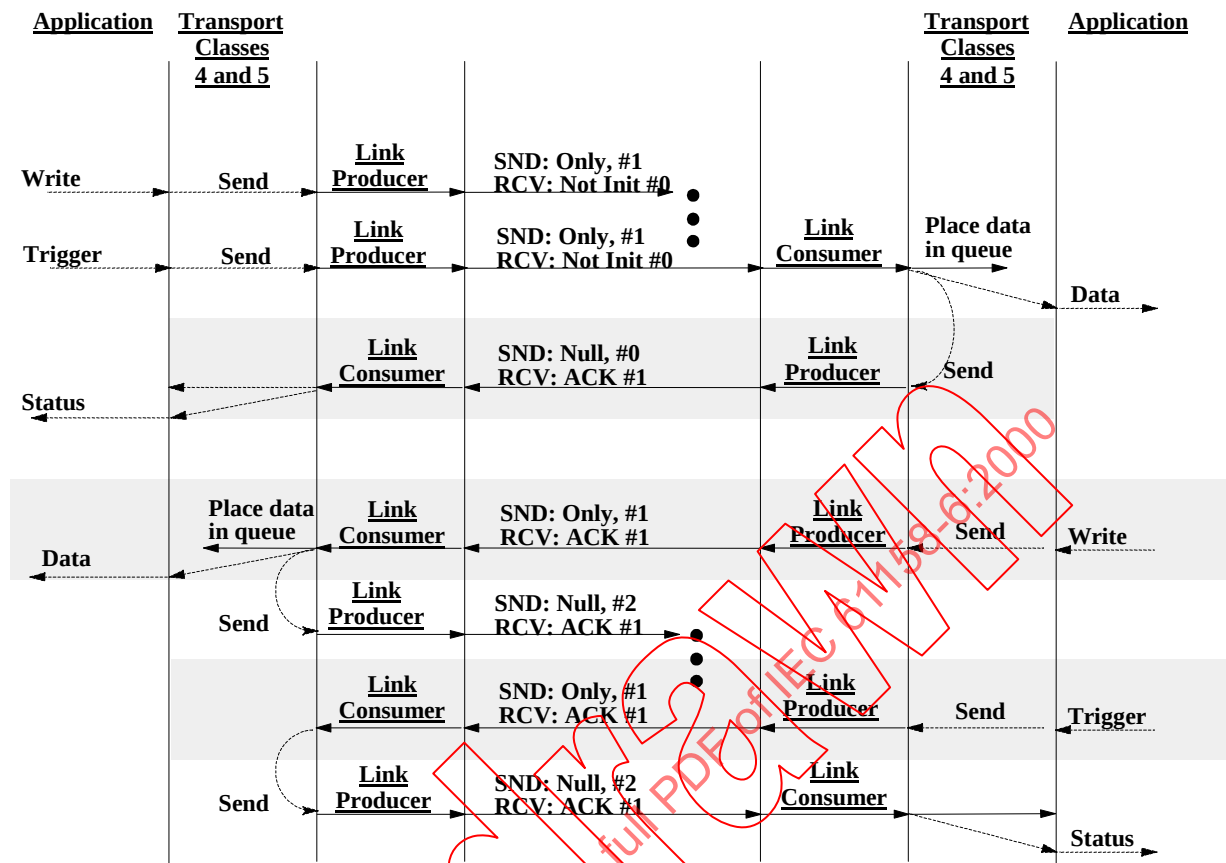


Figure 86: Sequence Diagram of Retries Using Transport Classes 4 and 5

5.6.2.9.2.6 Typical Idle Sequence (Informative)

When a class 4 or 5 transport has nothing to send which shall impact the other state machine, it does not need to send anything. However, it is necessary to send some small amount of traffic to keep the connection alive. The Trigger event shall be used, with the timer set to a “keep alive” value. Figure 87 shows the *Null* command used to keep the connection alive, using the Trigger event.

Note, the *Null* command does not need to be acknowledged. The other transport shall not retry a *Null* command. It shall continue to resend the *Null* command at the “keep alive” rate.



To make the specification of the Non-Blocking classes (Classes 4 and 5) easier to read, the specification describes them as being made up of two state machines: one to handle the “Sender” function (Transport PDU’s from this device to the other) and the other to handle the “Receiver” function (Transport PDU’s from the other device to this one). (The class 6 transport has a client and a server state machine similar to the class 3 transport.)

The Sender state machine has control of the Link Producer; the Receiver state machine accepts the data from the Link Consumer. The Sender state machine accepts data from the application; the Receiver state machine delivers data to the application. (This application data might be either Application Requests or Responses or data; the transport does not care nor know which it is.)

Figure 89 shows the basic structure for class 6.

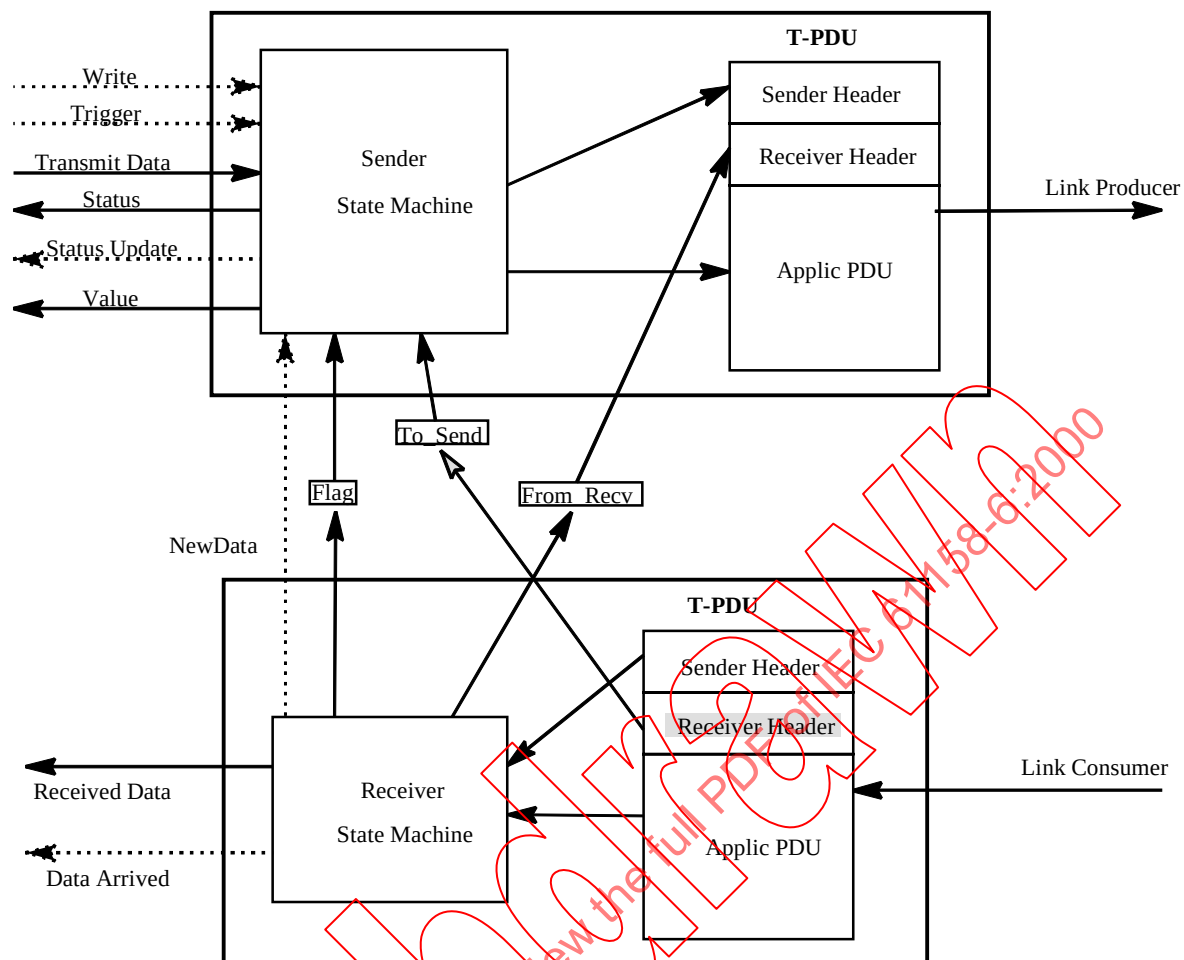


Figure 88. Classes 4 and 5 basic structure

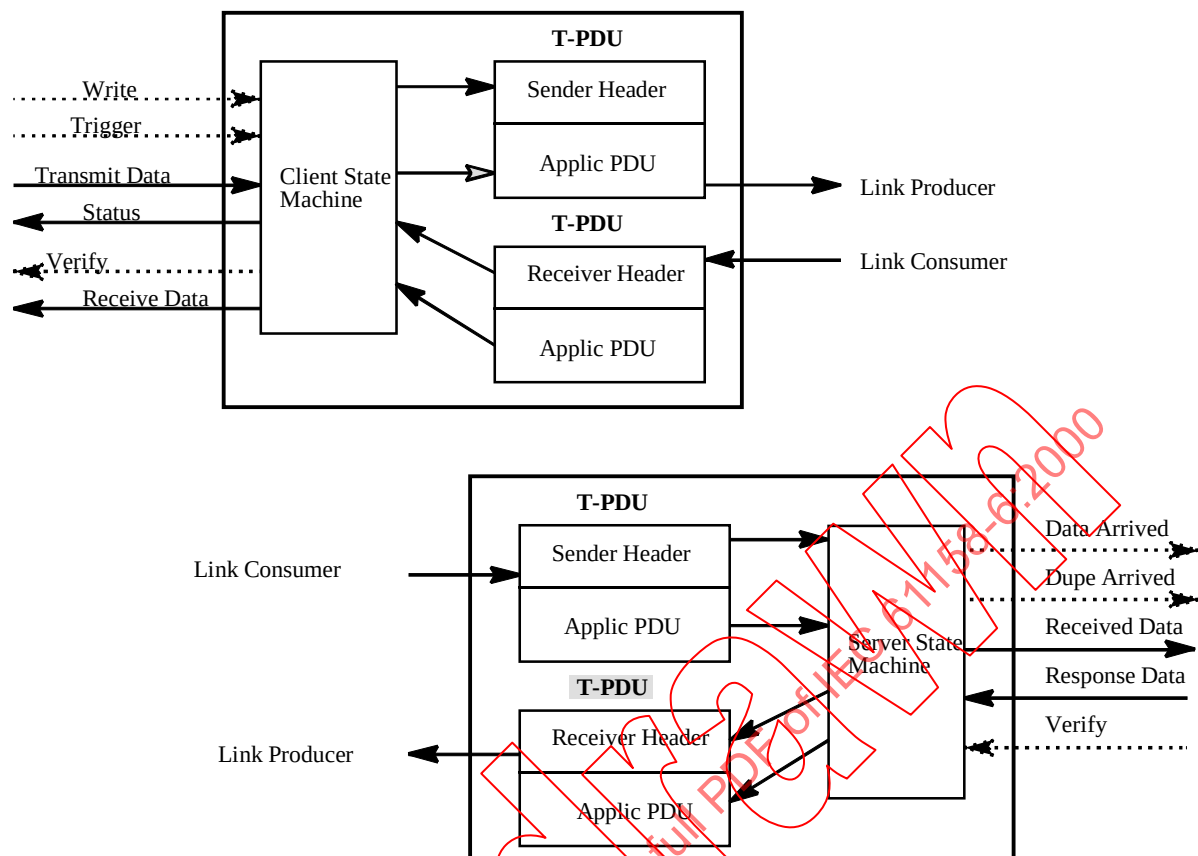


Figure 89. Class 6 basic structure

5.6.2.9.4 Classes 4 to 6 General States

The defined states and their descriptions of the Class 4 Transport Sender are listed in Table 317 below :

Table 317: Common States for Transport Classes 4 to 6

State	Description
Non-existent	The instance of Transport Class 4 to 6 does not exist.
Idle	The instance of Transport Class 4 to 6 has been created but not yet started.
Operational	The instance of Transport Class 4 to 6 has been created and started. Note : this state has sub-states described in features particular to individual classes 4, 5 and 6.

5.6.2.9.5 Classes 4 to 6 General State Transitions

Figure 90 shows the general behaviour of the transport classes. Each of the classes has some additional details added to the “Operational” state. This general behaviour shall **not** be repeated for each class in the interest of emphasizing the common behaviours and allowing those other descriptions to focus on the unique aspects of that class.

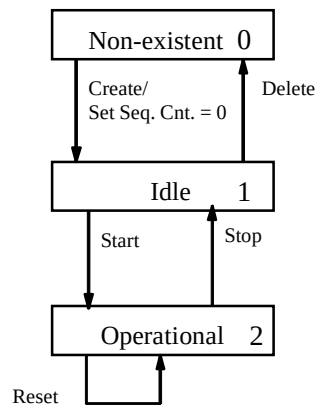


Figure 90: Classes 4 to 6 general STD

5.6.2.9.6 Classes 4 to 6 General State Event Matrix

Table 318 defines the general state transitions of the transport classes.

Table 318: Classes 4 to 6 general SEM

Event	State		
	Non-existent	Idle	Operational
Create	1) Seq. Cnt. = 0 2) Transition to Idle	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Operational	Error
Stop	Error	No Action	Transition to Idle
Reset	Error	Error	Generally, stop waiting for outstanding events; details are transport class specific

5.6.2.9.7 Production Triggers

IEC 61158-6 defines how these transport classes act as different trigger modes are specified when the Connection Manager creates an instance of the transport.

5.6.2.10 Transport State Machines – Class 4

5.6.2.10.1 Functions

The sequence count is set to 0 by the create service. It is incremented by the transport to distinguish a new Transport PDU from a retransmission of a previous one. It has a maximum value of 15. After that it rolls over to 0.

Transport class 4 is similar to having two class 2 transports, one in each direction, except that the class 4 uses two Network Connections instead of four. Transport class 4 can transfer application information in both directions. The delivery is acknowledged by the receiving transport. It uses a sequence number to identify which T-PDU is being acknowledged. Retries are performed.

The main advantage of this class over using two Class 2 (Acknowledgement) transports is that class 4 uses fewer resources. The main advantage of class 4 over class 3 is that it does not block the application from sending a second request before the first response is received. This pipelining can improve the system performance in some circumstances. In addition, class 4 supports bi-directional, asynchronous data transmission, without imposing request-response type serialisation.

This transport class notifies the application that the server application has responded to the transmitted packet. This transport class shall automatically send the next item in the queue when it receives an acknowledge that the previous one has been acknowledged. The application shall be notified if the network connection fails.

Transport class 4 uses two connections: the originator-to-target connection, which shall be point-to-point; and the return connection, which also shall be point-to-point.

The receiving transport initiates and sends the acknowledgement *after* the consuming node of the network connection has written the packet to the T-PDU buffer, *after* the transport has read the packet's transport header, and *after* the transport has placed a copy of the data into a queue for the application. The receiver's acknowledgement signals the sending transport that the receiver-side T-PDU receive buffer is ready to accept the next data packet.

The limit to the number of application requests which can be sent before an application response is received is determined by the design of the target application. It shall typically be limited by the amount of memory which the receiving transport can allocate for holding the outstanding messages. An error shall be returned to the sending transport when a transmission attempts to exceed the limit.

5.6.2.10.2 Transport Class 4 Sender

5.6.2.10.2.1 Functions

The class 4 Sender transport is responsible for:

- Accepting the application's *write* that indicates the application has written a data packet to the transmit data queue;
- Writing the application data and the transport headers to the Sender T-PDU transmit buffer for each packet that the application writes to it (NOTE: The transport header for each packet shall include a sequence count.);
- Initialising the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection;
- Performing retries if necessary when *triggered*;
- Accepting notification from the associated Receiver state machine that it has written the Receiver header (the other transport instance's verification) to the shared data area;
- Notifying the application that the application data has or has not been sent.

The following internal storage is assumed:

- Current Sequence Number (SN)
- Queue length
- Try counter

5.6.2.10.2.2 States

The defined states and their descriptions of the Class 4 Transport Sender are listed in Table 319 below:

Table 319: Class 4 Transport Sender States

State	Description
Ready	The instance of Transport Class 4 has been created and started. Queue is empty.
Waiting	The instance of Transport Class 4 has been created and started. Queue is not empty.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

5.6.2.10.2.3 State Transitions

State transitions of the class 4 sender transport are shown in Figure 91.

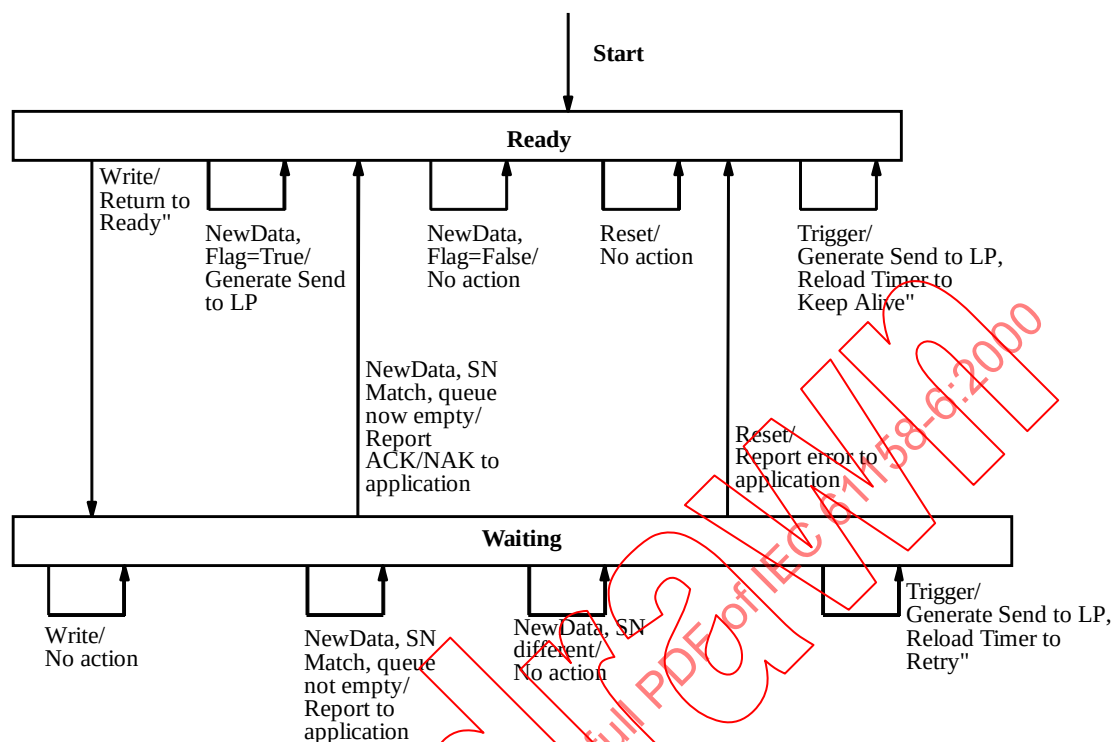


Figure 91: Class 4 sender STD

NOTE This STD is incomplete. It shows the most common transitions. See Table 320 for the full specification of all events in all conditions.

The Command and Sequence Number columns in Table 320 refer to the values of those fields of the *To_Send* data item. In the action descriptions, the phrase "set CMD" refers to setting the value of the Command field of the Sender Header in the T-PDU which shall be delivered to the Link Producer at the next *Send* event.

On all occurrences of the *New Data* event, the contents of the *From_Recv* data item are copied to the Receiver Header in the T-PDU.

NOTE This action is not listed in Table 320.

5.6.2.10.2.4 State Event Matrix

State transitions of the class 4 sender transport are shown in Table 320.

Table 320: Class 4 sender SEM

Event	Command	Sequence Number	State	
			Ready	Waiting
Reset	Don't care	Don't care	Do "return to Ready processing"	Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do "return to Ready processing"
Write	Don't care	Don't care	Do "return to Ready processing"	Ignore (Shall be recognised as part of "return to Ready processing")
NewData	ACK	Different	(Not expecting anything). Do "Flag processing".	(Assume the other transport is ACKing something old) Do "Flag processing".
		Same		Take the application data off the queue, place Success in Status queue, generate Status Updated event to application, do "return to Ready processing"
	NAK	Different		(Assume the other transport is NAKing something old) Do "Flag processing".
		Same		Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do "return to Ready processing"
	Other	Don't care		(Assume other end is confused.) Do "Flag processing".
Trigger (Retry or Keep alive timer)	Don't care	Don't care	Generate Send event to Link Producer (which re-sends the <i>Null</i> message to keep connection alive), reload timer with "keep alive" value.	Decrement Try counter, if zero: Take the application data off the queue, report error to application, generate Can't Send event to application, do "return to Ready processing" else: Generate Send event to Link Producer (which re-sends the <i>Only</i> message), reload timer with retry value.

The "Return to Ready Processing" includes the following:

- If the queue is now empty:
Increment Sequence number and place in Sender Header of T-PDU
Set CMD to *Null*
Load timer with "keep alive" value
Do the "Flag Processing" (described below)
Go to Ready state
- If the queue is not empty:
Increment Sequence number and place in Sender Header of T-PDU
Set CMD to *Only*
Set Try counter
Copy message to T-PDU buffer
Generate Send event to Link Producer
Load timer with "retry" value
Go to Waiting state

The "Flag Processing" is defined as: Test the value of the Flag data item from the Receiver state machine. If the Flag has a value of True then generate the Send event to the Link Producer. If the Flag has a value of False, do nothing.

5.6.2.10.3 Transport Class 4 Receiver

5.6.2.10.3.1 Functions

In the link to application direction, a class 4 Receiver transport is responsible for:

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU receive buffer;
- Locating and reading the transport headers of each packet that the consuming node of the network connection writes to the server-side T-PDU receive buffer;
- Performing the actions specified in the state event matrix for each packet which arrives;
- Notifying the server application when it places new data into the received data queue;
- Deliver the appropriate data (*To_Send*, *From_Recv*, and *Flag*) to the associated Sender transport state machine with the *NewData* event.

The Receiver Bi-directional Point-to-Point Non-Blocking transport has two states:

- Initial
- Ready

The basic approach is to start in the *Initial* state and transition to the *Ready* state when it receives something that indicates the receiver is synchronised with the sender. It remains in the *Ready* state until the *Reset* service is applied.

5.6.2.10.3.2 States

The defined states and their descriptions of the Class 4 Transport Receiver are listed in Table 321 below :

Table 321: Class 4 Transport Receiver States

State	Description
Ready	The instance of Transport Class 4 has been created and started. Queue is empty.
Waiting	The instance of Transport Class 4 has been created and started. Queue is not empty.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

5.6.2.10.3.3 State Transitions

State transitions of the class 4 receiver transport are shown in Figure 92.

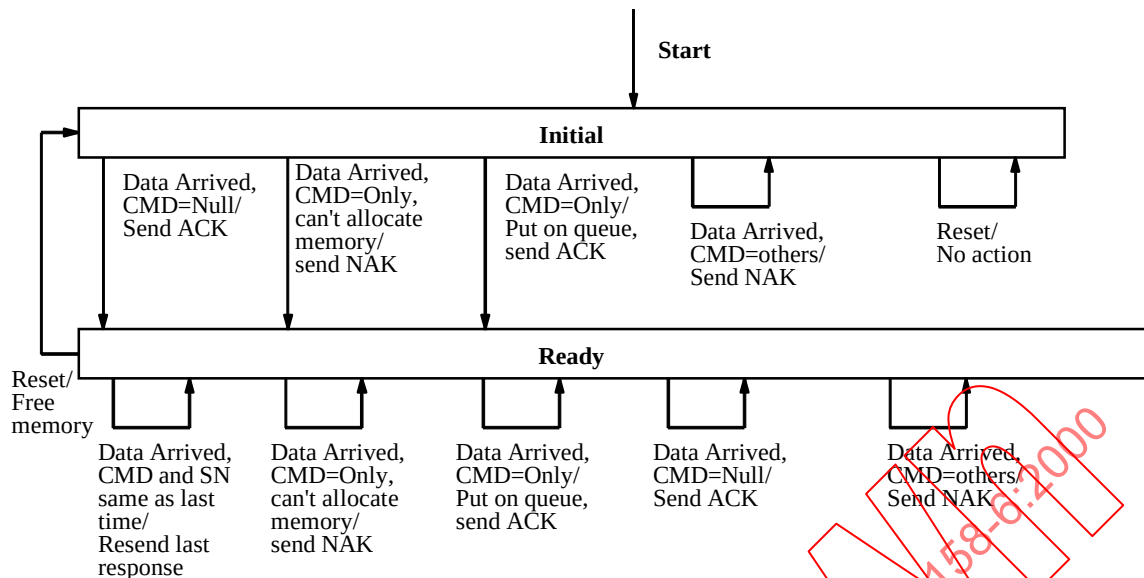


Figure 92: Class 4 receiver STD

The following events can happen. Some of these are labelled on the Data Flow Diagram as the Packet Arrived event; I have taken the liberty to also factor in what arrived and if it has the same or different Sequence Number (SN):

- Data Arrived, CMD = Null;
- Data Arrived, CMD = Only;
- Data Arrived, CMD is Illegal;
- Reset;

NOTE There is **no** timer event used by the Receiver state machine. (If there is a communications problem, the Sender state machine shall time out. The application shall have a timer to indicate problems with the other application).

On all occurrences of the *Data Arrived* event, the contents of the Receiver Header are copied to the *To_Send* data item.

NOTE This action is not listed in Table 322.

The phrase "Set CMD" used in Table 322 means "set the Command field of the From_Recv data item". When the value of CMD or SN is used to make a decision, it refers to the value of the Command or Sequence Number fields of the Sender Header in the T-PDU received from the Link Consumer which caused the Data Arrived event.

The following internal storage is assumed:

- Previous SN
- Previous CMD received

5.6.2.10.3.4 State Event Matrix

State transitions of the class 4 receiver transport are shown in Table 322.

Table 322: Class 4 receiver SEM

Event	State	
	Initial	Ready
Reset	No action	Set the Command field of the <i>From_Recv</i> data item to <i>Not_Init</i> and the Sequence Number of the <i>From_Recv</i> data item to zero. Go to Initial.
Data Arrived, CMD=Null, SN different	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate <i>NewData</i> event, go to Ready.	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate <i>NewData</i> event, stay in Ready.
Data Arrived, CMD=Null, SN same		If previous CMD was <i>Null</i> , then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Set Flag to False, generate <i>NewData</i> event, stay in Ready. (There is no need to rush out an ACK to a <i>Null</i> , let the sender's refresh send it out.) else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to False, generate <i>NewData</i> event, stay in Ready.
Data Arrived, CMD=Only, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate <i>NewData</i> event, go to Ready. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate <i>NewData</i> event, go to Ready.	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate <i>NewData</i> event, stay in Ready. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate <i>NewData</i> event, stay in Ready.
Data Arrived, CMD=Only, SN same		If previous CMD was <i>Only</i> , then: Set Flag to True, generate <i>NewData</i> event, stay in Ready. (This is a retry, assume SN and CMD are OK.) (Flag is TRUE to speed recovery from noise, at the possible expense of some additional link transmit time.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate <i>NewData</i> event, stay in Ready.
Data Arrived, CMD=Illegal	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate <i>NewData</i> event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate <i>NewData</i> event, stay in Ready.

5.6.2.11 Transport State Machines – Class 5

5.6.2.11.1 Functions

Possible uses of transport class 5 include exchanging messages and responses up to 65536 bytes between two applications.

The sequence count is set to 0 by the create service. It is incremented by the transport to distinguish a new Transport PDU from a retransmission of a previous one. It has a maximum value of 15. After that it rolls over to 0 limiting the transport to fewer than 16 outstanding messages.

Transport class 5 is similar to class 4, except that class 5 adds Fragmentation and Reassembly at the transport layer. Transport class 5 can transfer application information in both directions. The delivery of each fragment is acknowledged by the receiving transport. A sequence number identifies which T-PDU is being acknowledged. Retries are performed.

This transport class notifies the application that the server application has responded to the transmitted packet. This transport class shall automatically send the next item in the queue when it receives an acknowledgement that the previous one has been acknowledged. The application shall be notified if the network connection fails.

Transport Class 5 uses two connections: the originator-to-target connection, which shall be point-to-point, and the return connection, which also shall be point-to-point.

The receiving transport initiates and sends the acknowledgement *after* the consuming node of the network connection has written the packet to the T-PDU buffer, *after* the transport has read the packet's transport header, and *after* the transport has placed a copy of the data into a queue for the application. The receiver's acknowledgement signals the sending transport that the receiver-side T-PDU receive buffer is ready to accept the next data packet.

The limit to the number of application requests which can be sent before an application response is received is determined by the design of the target application. It shall typically be limited by the amount of memory. An error shall be returned when a transmission attempts to exceed the limit.

5.6.2.11.2 Class 5 Sequence Diagrams (Informative)

5.6.2.11.2.1 Typical Sequences (Informative)

Some typical sequences are illustrated to aid understanding of the operation of this transport class. A formal specification of this transport class is given in IEC 61158-5.

NOTE The following sequences are identical for classes 4 and class 5. They are described under the class 4 specification and are not repeated here.

- Typical Message Exchange Sequence (see 5.6.2.9.2.2);
- Typical Sequence with Messages overwriting each other (see 5.6.2.9.2.3);
- Typical Queued Sequence (see 5.6.2.9.2.4);
- Typical Sequence with retries (see 5.6.2.9.2.5);
- Typical Idle Sequence (see 5.6.2.9.2.6).

5.6.2.11.2.2 Typical Three Fragment Sequence (Informative)

Figure 93 shows a typical sequence where the application data is too large to fit into one T-PDU. In the example shown, the application data is split into three fragments. In general, there could be many *Middle* T-PDUs. Each *Middle* T-PDU increments the Sequence Number in the Sender Header. The *Last* T-PDU shall also increment the Sequence Number.

At the receiver, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

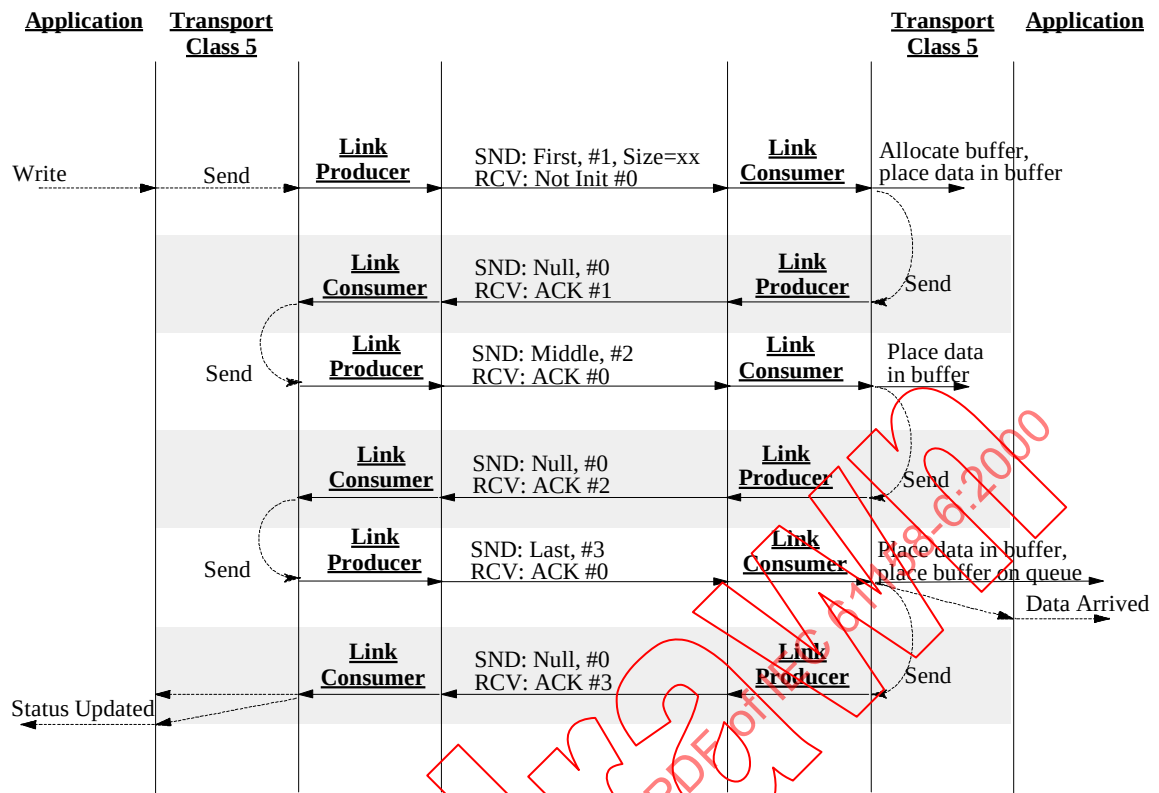


Figure 93: Sequence Diagram of Three Fragments Using Transport Class 5

5.6.2.11.2.3 Typical Fragmented Sequence With Retries (Informative)

Figure 94 shows the processing when a transmission error causes something to be lost. In this example, two transmissions are lost, one in each direction.

NOTE The processing is very similar to class 4 (see 5.6.2.9.2.5).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

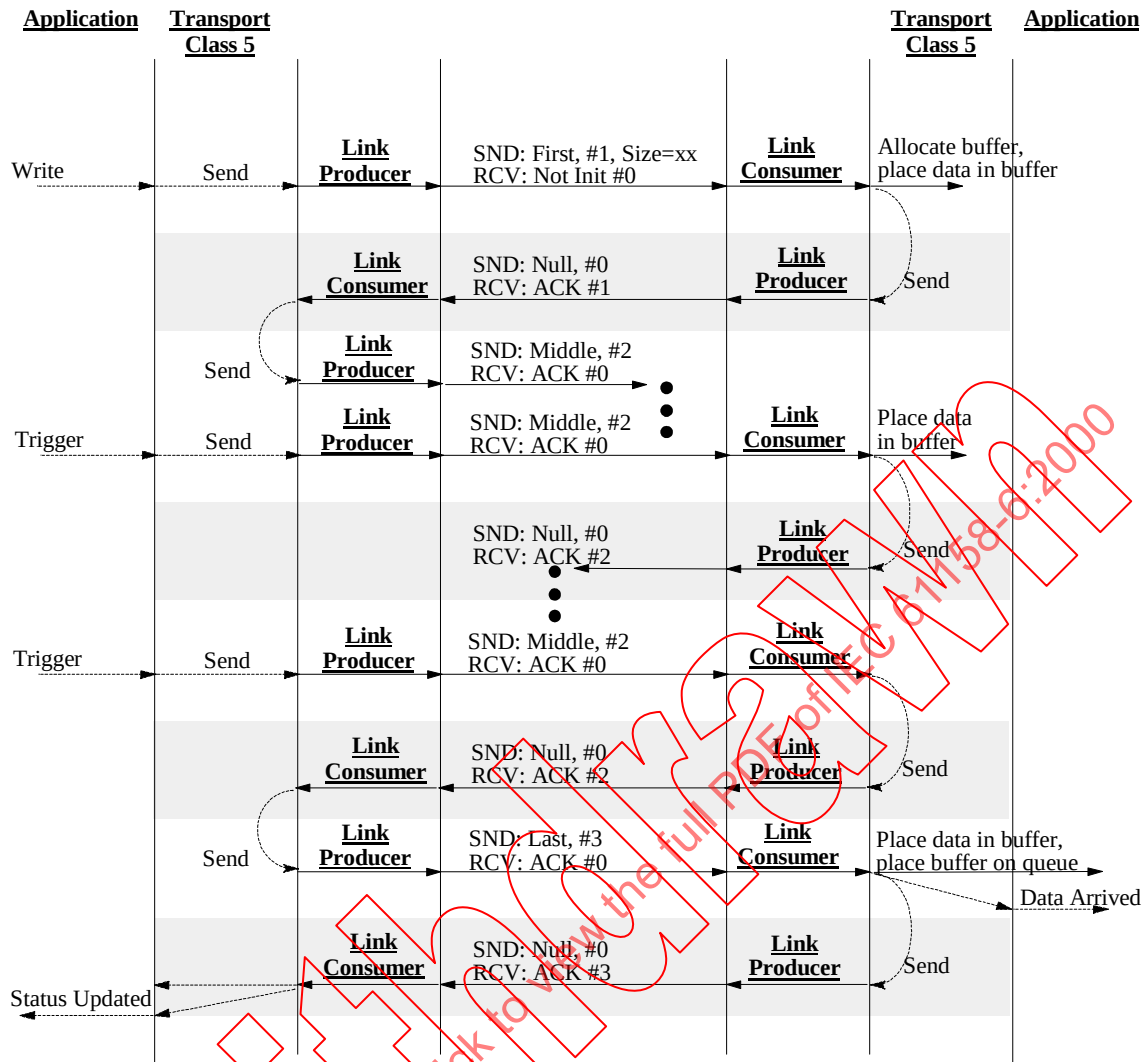


Figure 94: Sequence Diagram of Fragmentation with Retries Using Transport Class 5

5.6.2.11.2.4 Typical Two Fragment Sequence (Informative)

Figure 95 shows a special case, where there are only two fragments. In this case, there is no *Middle* T-PDU. Again, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

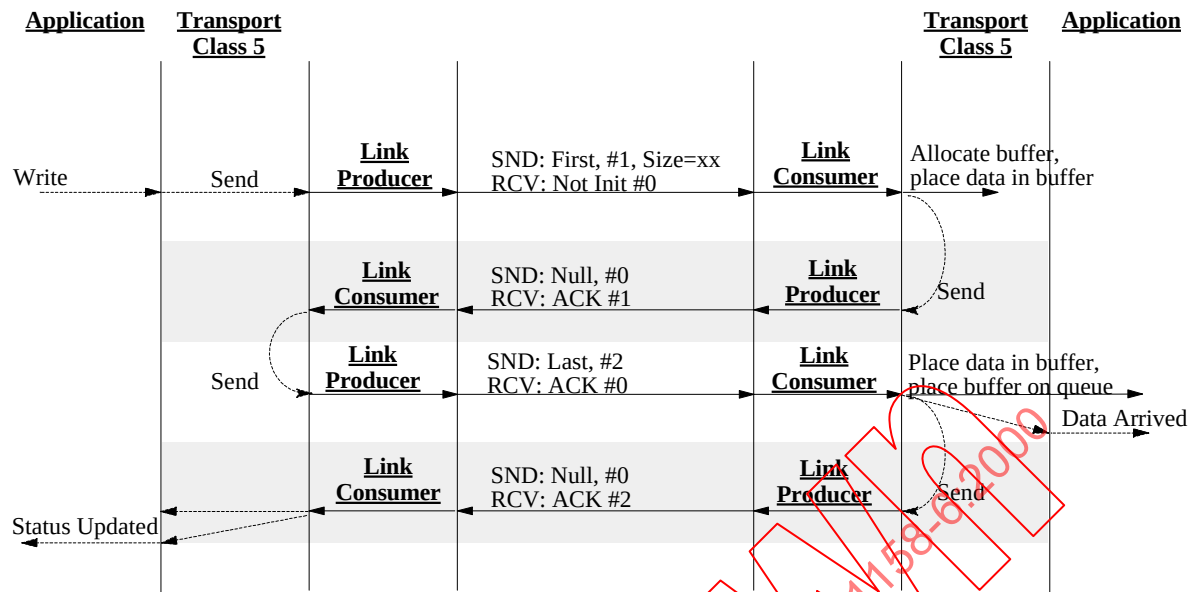


Figure 95: Sequence Diagram of Two Fragments Using Transport Class 5

5.6.2.11.2.5 Typical Aborted Fragmented Sequence (Informative)

Figure 96 shows a receiver recovering when the sender has been reset in the middle of a fragmented sequence. When the receiver gets a *First* or *Only*, it assumes that any message which did not receive a *Last* has been aborted.

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

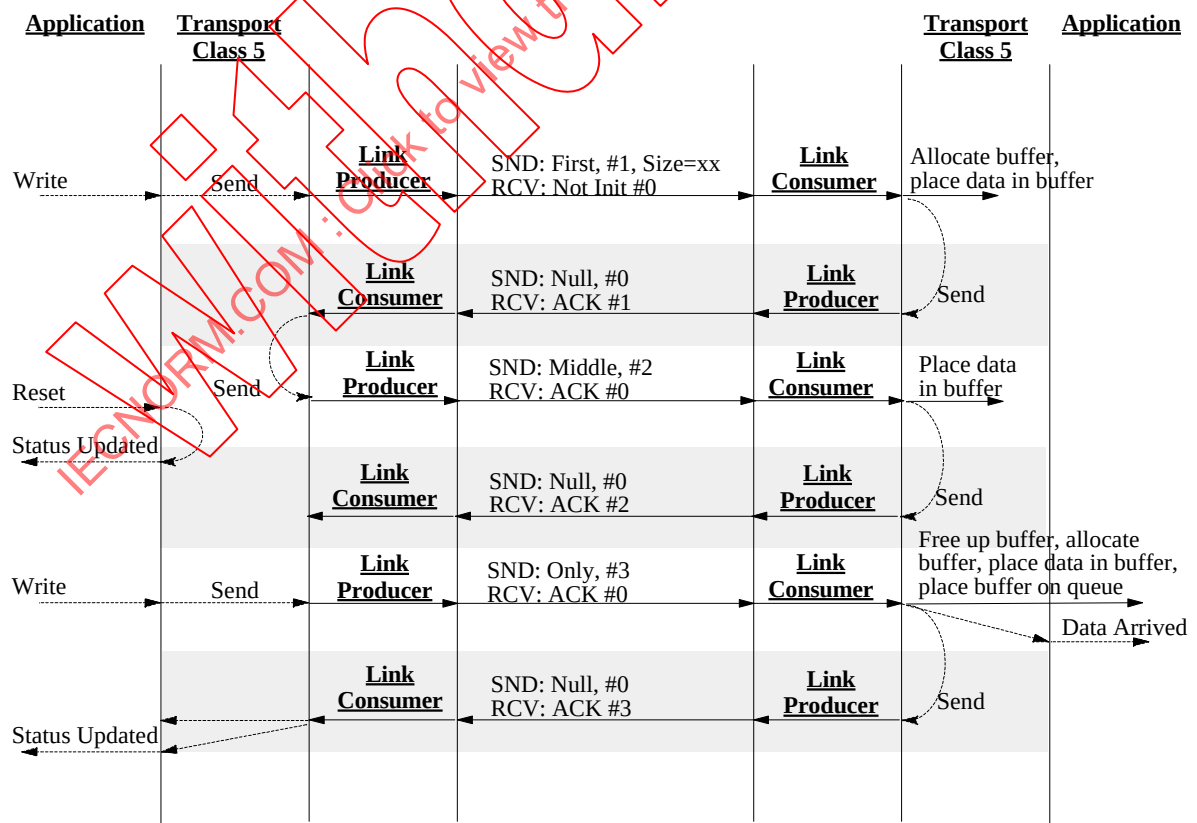


Figure 96: Sequence Diagram of Aborted Message Using Transport Class 5

5.6.2.11.3 Transport Class 5 Sender

5.6.2.11.3.1 Functions

The Class 5 Sender transport is responsible for:

- Accepting the application's *write* that it has written a data packet to the transmit data queue;
- Writing the application data and the transport headers to the Sender T-PDU transmit buffer for each packet that the application writes to it;
- Initialising the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection;
- Performing retries if necessary when *triggered*;
- Accepting notification from the associated Receiver state machine that it has written the Receiver header (the other transport instance's verification) to the shared data area;
- Notifying the application that the application data has or has not been sent.

The following internal storage is assumed:

- Current Sequence Number (SN);
- Queue length;
- Try counter.

5.6.2.11.3.2 States

The defined states and their descriptions of the Class 5 Transport Sender are listed in Table 323 below :

Table 323: Class 5 Transport Sender States

State	Description
Ready	The instance of Transport Class 5 has been created and started. Queue is empty.
Waiting	The instance of Transport Class 5 has been created and started. Queue is not empty.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

5.6.2.11.3.3 State Transitions

State transitions of the class 5 sender transport are shown in Figure 97.

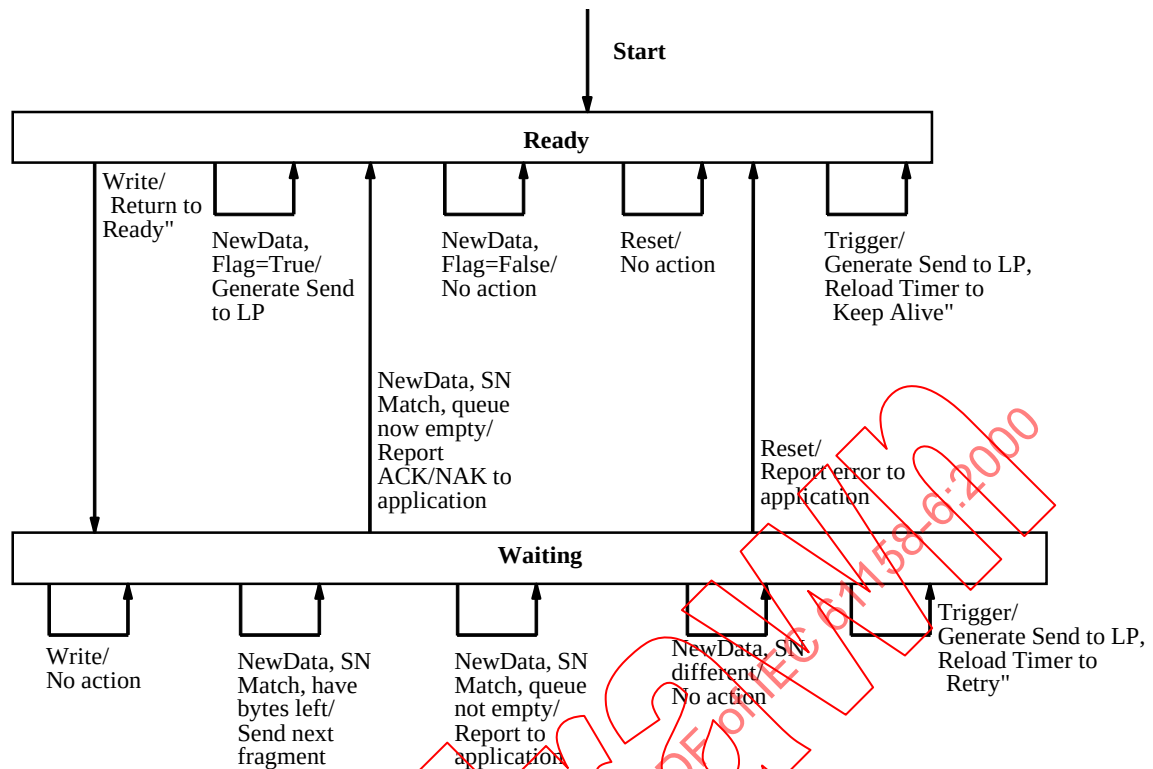


Figure 97: Class 5 sender STD

NOTE This STD is incomplete. It shows some common transitions. See Table 324 for the full specification of all events in all conditions.

The Command and Seq. Number columns in Table 324 refer to the values of those fields of the *To_Send* data item. In the action descriptions, the phrase "set CMD" refers to setting the value of the Command field of the Sender Header in the T-PDU which shall be delivered to the Link Producer at the next *Send* event. Also, the phrase "increment sequence number" refers to setting the value of the Sequence Number field of the Sender Header in the T-PDU which shall be delivered to the Link Producer at the next *Send* event.

On all occurrences of the *New Data* event, the contents of the *From_Recv* data item are copied to the Receiver Header in the T-PDU.

NOTE This action is not listed in Table 324.

5.6.2.11.3.4 State Event Matrix

State transitions of the class 5 sender transport are shown in Table 324.

Table 324: Class 5 sender SEM

Event	Command	Sequence Number	State	
			Ready	Waiting
Reset	n/a	n/a	Do "return to Ready processing"	Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do "return to Ready processing"
Write	n/a	n/a	Do "return to Ready processing"	Ignore (Shall be recognised as part of "return to Ready processing")
NewData	ACK	Different	(Not expecting anything). Do "Flag processing".	(Assume the other transport is ACKing something old) Do "Flag processing".
		Same		If the whole application message has been sent, then take the application data off the queue, place Success in Status queue, generate Status Updated event to application, do "return to Ready processing" else, if the remainder fits in one fragment, then place the remainder in the Link Producer buffer, increment the sequence count, set the CMD to <i>Last</i> , and generate the Send event to the Link Producer else, place the next fragment in the Link Producer buffer, increment the sequence count, set the CMD to <i>Middle</i> , and generate the Send event to the Link Producer and stay in Waiting state.
	NAK	Different		(Assume the other transport is NAKing something old) Do "Flag processing".
		Same		Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do "return to Ready processing"
	Other	n/a		(Assume other end is confused.) Do "Flag processing".
Trigger (Retry or Keep alive timer)	n/a	n/a	Generate Send event to Link Producer (which re-sends the <i>Null</i> message to keep connection alive), reload timer with "keep alive" value.	Decrement Try counter, if zero: Take the application data off the queue, report error to application, generate Can't Send event to application, do "return to Ready processing" else: Generate Send event to Link Producer (which re-sends the previous message), reload timer with retry value.

The "Return to Ready Processing" includes the following:

- If the queue is now empty:
 - Increment Sequence number
 - Set CMD to *Null*
 - Load timer with "Keep alive" value
 - Do the "Flag Processing" (described below)
 - Go to Ready state
- If the queue is not empty:
 - Increment Sequence number
 - Test if the message shall fit in one fragment; if it does:

- Set CMD to *Only*
 - Copy message to T-PDU buffer
- else
- Set CMD to *First*
 - Calculate (total size / 256 bytes) - 1; put into Sender Header
 - Copy first fragment to T-PDU buffer

Set Try counter

Load timer with “Retry” value

Generate Send event to Link Producer

Go to Waiting state

The “Flag Processing” is defined as: Test the value of the Flag data item from the Receiver state machine. If the Flag has a value of True then generate the Send event to the Link Producer. If the Flag has a value of False, do nothing.

5.6.2.11.4 Transport Class 5 Receiver

5.6.2.11.4.1 Functions

In the link to application direction, a Class 5 Receiver transport is responsible for:

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU receive buffer;
- Locating and reading the transport headers of each packet that the consuming node of the network connection writes to the server-side T-PDU receive buffer;
- Performing the actions specified in the state event matrix for each packet which arrives;
- Notifying the server application when it places new data into the received data queue;
- Deliver the appropriate data and flags to the associated Sender transport.

The Receiver Bi-directional Point-to-Point Non-Blocking transport has three states:

- Initial;
- Assembling;
- Ready.

The following internal storage is assumed:

- Previous Sequence Number (SN);
- Previous Command (CMD).

The basic approach is to start in the *Initial* state and transition to either the *Assembling* or the *Ready* state when it receives something that indicates the receiver is synchronised with the sender. It then stays in the *Ready* and *Assembling* states until the *Reset* service is applied. It goes into the *Assembling* state after receiving a *First* T-PDU and remains there until getting the *Last* T-PDU (unless it thinks the sender has aborted sending the current T-SDU).

5.6.2.11.4.2 States

The defined states and their descriptions of the Class 5 Transport Receiver are listed in Table 325 below :

Table 325: Class 5 Transport Receiver States

State	Description
Initial	The instance of Transport Class 5 has been created and started.
Assembling	The instance of Transport Class 5 has been created and started. Segmented data is being assembled.
Ready	The instance of Transport Class 5 has been created and started. Segmented data has been assembled.

NOTE These three states are sub-states of the Running state described as the common feature for classes 4 to 6.

5.6.2.11.4.3 State Transitions

State transitions of the class 5 receiver transport is shown in Figure 98.

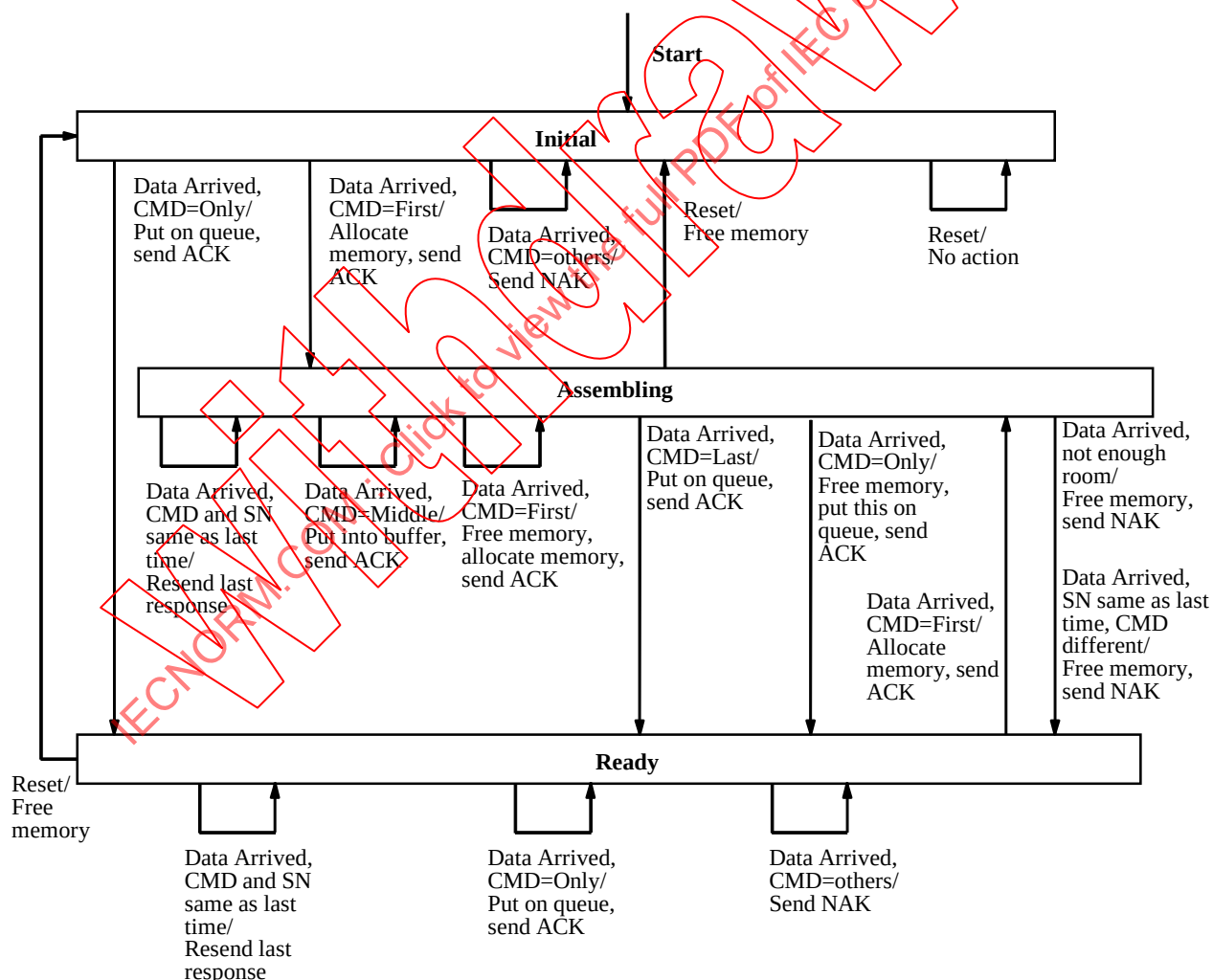


Figure 98: Class 5 receiver STD

The following events can happen.

- *Reset* Event;
- *Data Arrived* Event, *CMD= Null*;
- *Data Arrived* Event, *CMD= Only*;
- *Data Arrived* Event, *CMD= First*;
- *Data Arrived* Event, *CMD= Middle*;
- *Data Arrived* Event, *CMD= Last*;
- *Data Arrived* Event, *CMD= Illegal command*.

NOTE There is **no** timer event used by the Receiver state machine. (If there is a communications problem, the Sender state machine shall time out. The application shall have a timer to indicate problems with the other application.)

The following internal storage is assumed:

- Previous SN;
- Previous CMD received;
- Buffer Pointer;
- Room Left.

5.6.2.11.4.4 State Event Matrix

State transitions of the class 5 receiver transport is shown in Table 326.

Table 326: Class 5 receiver SEM

Event	State		
	Initial	Ready	Assembling
Reset	No action.	Go to Initial.	Free memory previously allocated. Go to Initial.
Data Arrived, CMD=Null, SN different	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate <i>NewData</i> event, go to Ready.	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate <i>NewData</i> event, stay in Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate <i>NewData</i> event, stay in Assembling.
Data Arrived, CMD=Null, SN same		If previous CMD was <i>Null</i>, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Set Flag to False, generate <i>NewData</i> event, stay in Ready. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to False, generate <i>NewData</i> event, stay in Ready.	If previous CMD was <i>Null</i>, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Set Flag to False, generate <i>NewData</i> event, stay in Assembling. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to False, generate <i>NewData</i> event, go to Ready.

Event	State		
	Initial	Ready	Assembling
Data Arrived, CMD=Only, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, stay in Initial else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, stay in Ready else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, stay in Ready.	Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, go to Ready. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Only, SN same		If previous CMD was <i>Only</i> , then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, assume SN and CMD are OK.) (Flag is TRUE to speed recovery from noise, at the possible expense of some additional link transmit time.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Assembling. (The sender changed the CMD but not the SN.)
Data Arrived, CMD=First, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i> , initialise Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, go to Assembling state. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i> , initialise Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, go to Assembling state. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, stay in Ready.	Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i> , initialise Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, stay in Assembling else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=First, SN same		If previous CMD was <i>First</i> , then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, previous attempt errored, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If previous CMD was <i>First</i> , then: Set Flag to True, generate NewData event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Assembling.

Event	State		
	Initial	Ready	Assembling
Data Arrived, CMD=Middle, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, copy SN to <i>From_Recv</i> , update Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, stay in Assembling. else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Middle, SN same			If previous CMD was <i>Middle</i> , then: Set Flag to False, generate NewData event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Last, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, put buffer on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, go to Ready. else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Last, SN same		If previous CMD was <i>Last</i> , then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If previous CMD was <i>Last</i> , then: Set Flag to False, generate NewData event, go to Ready. (This is a retry, assume SN and CMD are OK.) (Note: This shall happen: If the previous CMD is <i>Last</i> , then we shall be in the Ready state.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Assembling.

Event	State		
	Initial	Ready	Assembling
Data Arrived, CMD=Illegal	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Ready.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Assembling.

5.6.2.12 Transport State Machines – Class 6

5.6.2.12.1 Functions

Possible uses of transport class 6 include delivering event notification messages (up to 65536 bytes) to multiple servers.

Transport class 6 is similar to class 3. Transport class 6 can transfer application information in both directions. Class 6 supports multicasting in the Client to Server direction. Class 6 adds Fragmentation and Reassembly at the transport layer, but only in the Client to Server (multicast) direction.

The delivery of each fragment is acknowledged by the receiving transport. A sequence number identifies the T-PDU being acknowledged. Retries are performed.

The sequence count is set to 0 by the create service. It is incremented by the transport to distinguish a new Transport PDU from a retransmission of a previous one. It has a maximum value of 15. After that it rolls over to 0.

Class 6 is **NOT** a superset of class 5. Class 6 is blocking; that is, the next application request is not delivered until the current one is acknowledged. Functionally, class 6 shall be thought of as a fragmenting version of class 3, rather than as a multicasting version of class 5. However, class 6 reuses the transport header format from class 5, instead of the class 3 transport header.

This transport class notifies the application that the server application has responded to the transmitted packet. The application shall be notified if the network connection fails.

Transport class 6 uses multiple network connections: the Client-to-Server connection, which may be multicast; and the return connections (one per Server), which shall be point-to-point.

One significant difference between class 3 and class 6, which results from class 6 doing fragmentation: the “write” event causes the first (or only) fragment to be transmitted; the “trigger” event is only a timer function which is used to signal retries or keep-alive transmissions.

5.6.2.12.2 Class 6 Data Flow Diagram

Figure 99 shows the data flow diagram for class 6. Note that there is a separate T-SDU and T-PDU in the Client to Server direction. The client places the fragments into the T-PDU. The server assembles them from the T-PDU into the T-SDU. Since there is no fragmentation in the Server to Client direction, the Application places the data directly into the Server’s outgoing T-PDU (as is done in class 3).

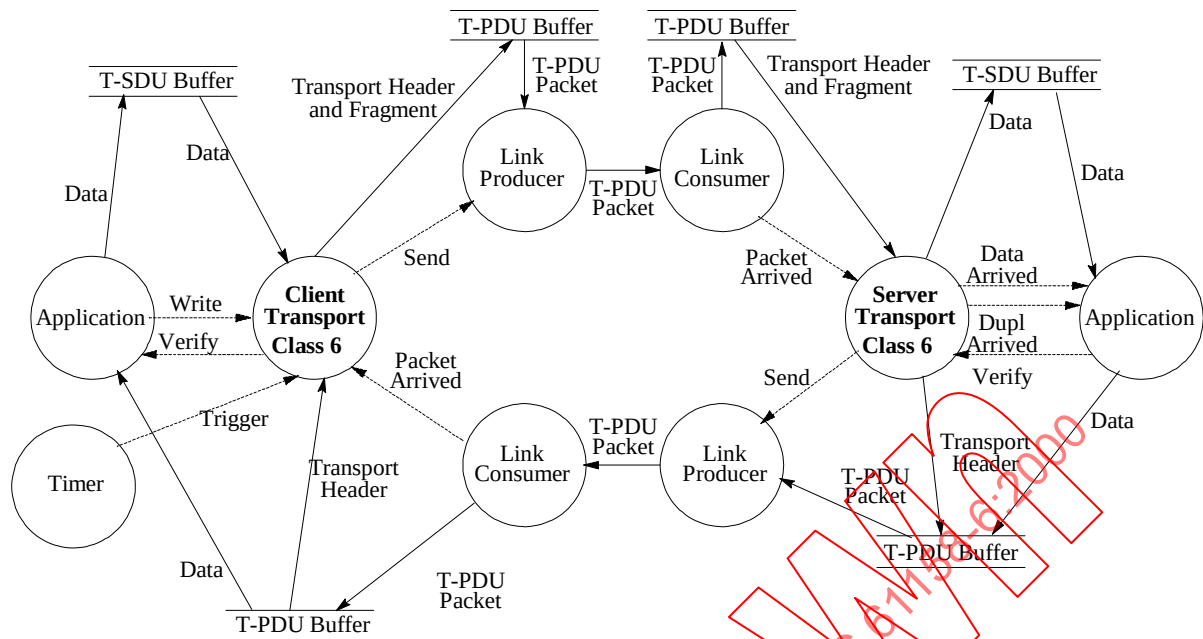


Figure 99: Data Flow Diagram For Transport Class 6

5.6.2.12.3 Class 5 Features Missing in Class 6

5.6.2.12.3.1 Function

It may be noted that the following two features of class 5 are not supported in class 6:

- The ability to have multiple outstanding requests queued between the receiving transport and its application;
- The support of fragmentation and reassembly in both directions.

This is a protocol design decision to exclude these features for the following reasoning. It is preferable that the class 6 transport header will not change size during the lifetime of a connection (due to the complexity of notifying all receivers of the change in header size). In the system, the multicasting receivers can enter and leave the relationship at will. Therefore, the number of multicasting receivers can change during the lifetime of the connection. Therefore, the size of the header shall be independent of the number of receivers. Since it is desirable to have no artificial limit to the number of receivers the decision was made to have **one** header field which all receivers use. This implies that all receivers shall be in the same state, which is maintained by having them all synchronised with the sender.

5.6.2.12.3.2 Blocking Versus Non-blocking

Assume there are several receivers of the multicast data. They are sending application responses to application requests to the multicasting sender. These application responses need to be acknowledged so that the transport can tell if it shall retry sending them or if it can now free up the resources used by that response. If the application responders are not synchronised with each other, they can be responding to different application requests. Therefore, each needs a field so that the application requester's transport can tell each of them about the successful delivery of the application response.

The design decision was not to have multiple fields, therefore all use the same field to decide if a retry is needed or if the resources can be released. Therefore, the transport delivery is synchronised with the application (the same as class 3).

5.6.2.12.3.3 Fragmentation in Both Directions

This was excluded by similar reasoning to the "Blocking vs. Non-Blocking" discussion. If the application responses from different responders require a different number of fragments, it would require the multicaster to send different transport headers to each of them.

5.6.2.12.4 Class 6 Sequence Diagrams (Informative)

5.6.2.12.4.1 Typical Sequences (Informative)

Some typical sequences unique to class 6 are illustrated to aid understanding of the operation of this transport class. A formal specification of this transport class is given in IEC 61158-5.

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

5.6.2.12.4.2 Typical Message Exchange Sequence (Informative)

Figure 100 shows the sequence in which actions take place during a messaging request/response exchange using transport class 6. The unshaded areas in Figure 100 indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction. Each transmission shows the Command and Sequence Number used for each header.

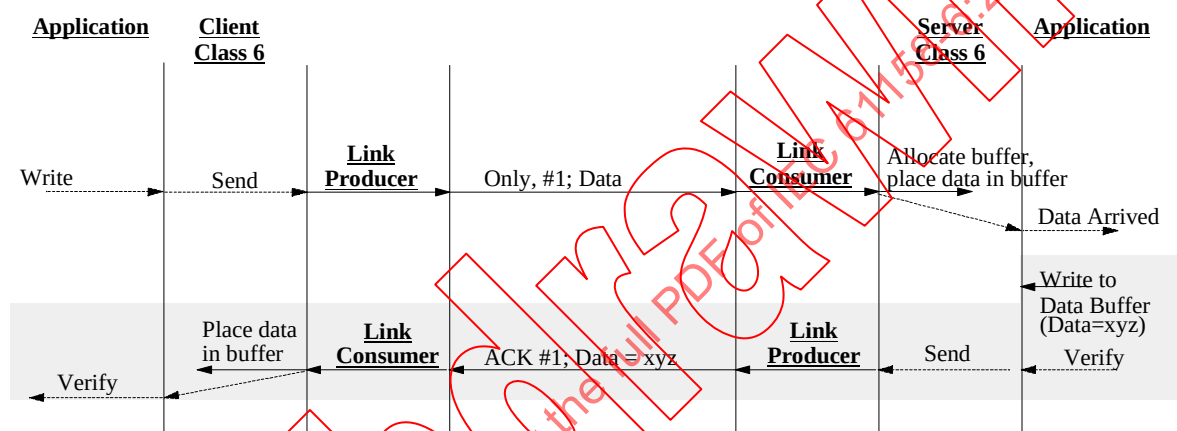


Figure 100: Sequence diagram of message exchange using transport class 6

5.6.2.12.4.3 Typical Message Exchange Sequence With Retries (Informative)

Retries are performed when a transport does not receive an Receiver header with the same sequence count as its current Sender header in the timeout period. The T-PDU is not changed for a retry.

Figure 101 shows two retry situations: where an *Only* command fails to be delivered and where an expected ACK fails to be delivered. This processing is very similar to class 3. The Trigger event is generated by the retry timer.

NOTE See 5.6.2.12.4.7 for a typical sequence with retries when the data is fragmented.

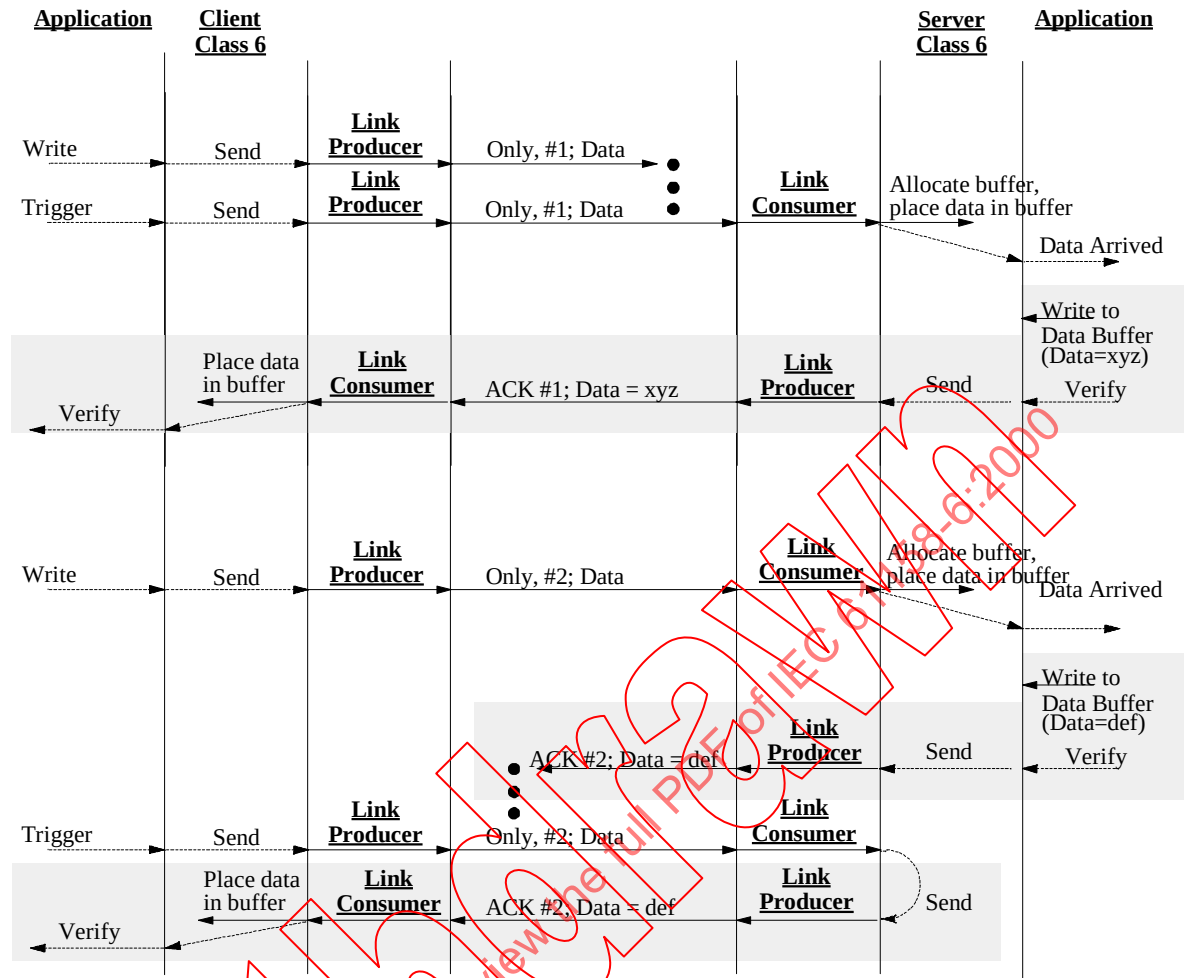


Figure 101: Sequence diagram of retries using transport class 6

5.6.2.12.4.4 Typical Idle Sequence (Informative)

When a class 6 Client transport has nothing to send which shall impact the other state machine, it does not need to send anything. However, it is necessary to send some small amount of traffic to keep the connection alive. The Trigger event shall be used, with the timer set to a “keep alive” value.

Figure 102 shows the *Null* command used to keep the connection alive, using the Trigger event.

Note, the Server does not have a timer, therefore the *Null* command shall be acknowledged by the server to keep the point to point network connection open. However, the client transport does not need to wait for the acknowledgement if the application gives it a Write event (see 5.6.2.12.4.5).

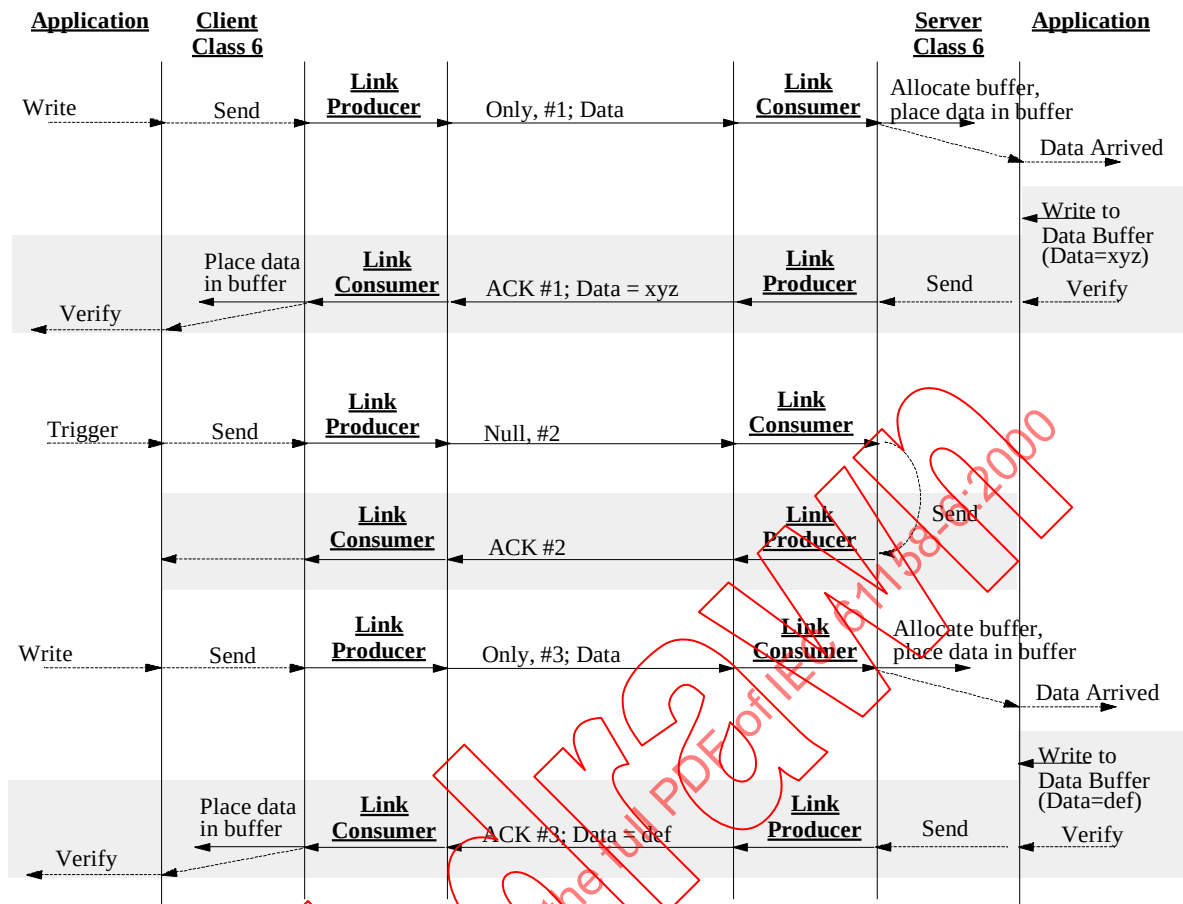


Figure 102: Sequence diagram of idle traffic using transport class 6

5.6.2.12.4.5 Typical Sequence With Overwrite (Informative)

It is permissible for a transport to send two data packets in a row. This implies that the second one might overwrite the first one in an intermediate module. This is desirable, since the second packet contains information that is newer than the first one and the data in the first one is no longer needed. There are two situations where this can occur when using Transport class 6.

In Figure 102 the transport on the left had nothing to send when it was triggered by the keep-alive timer. Therefore, it sent the *Null #2*. But when the application on the left did a *WRITE*, the transport immediately sent the *Only #3*. The previous example shows both of them arriving separately.

Figure 103 shows the left side's *Only #3* overwriting the *Null #2*.

Figure 104 shows a different situation where an overwrite can occur. The left side sends a *Null #2* as the result of the keep-alive *Trigger* event. But, before the *ACK #2* can be delivered, the right side receives the *Only #3* from the left side. The right side responded with the newer *ACK #3*, which overwrites the *ACK #2*.

The same situation shall occur if a *First* is used instead of the *Only*.

It is not necessary to wait for the ACK to a *Null* before sending an *Only* or a *First*.

When sending an *Only* or a *First*, it shall have a different sequence number than the *Null*, so sender knows which is being ACK'd by the other side.

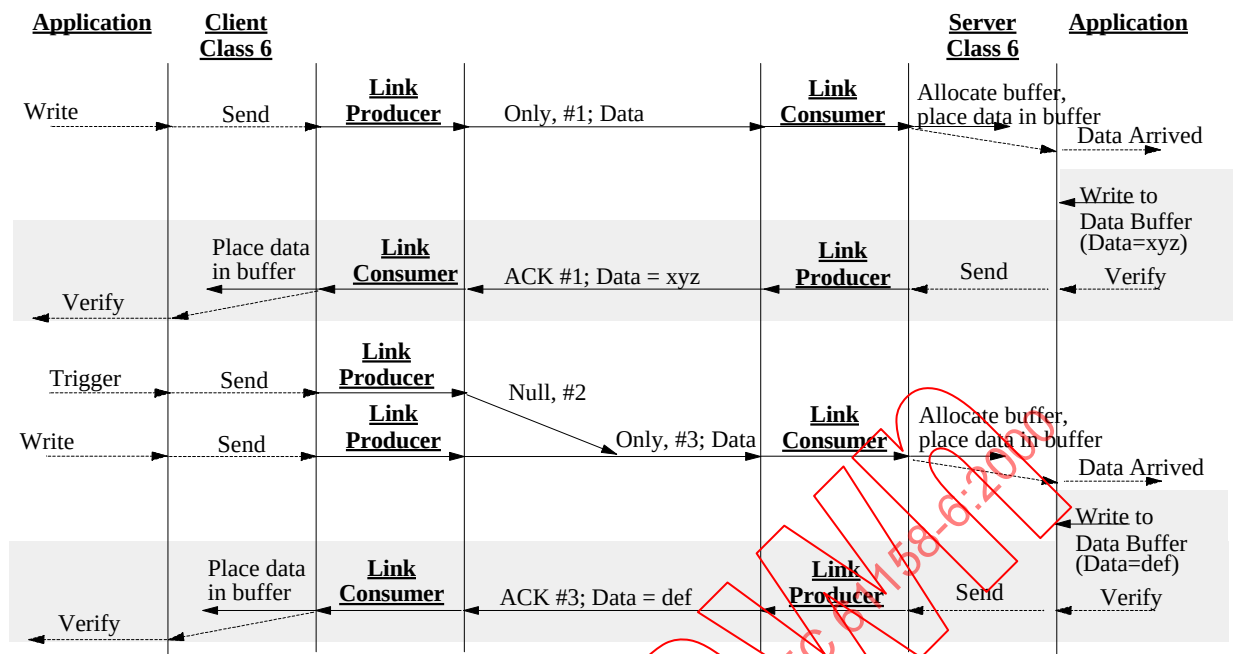


Figure 103: Sequence Diagram of Request Overwriting Null

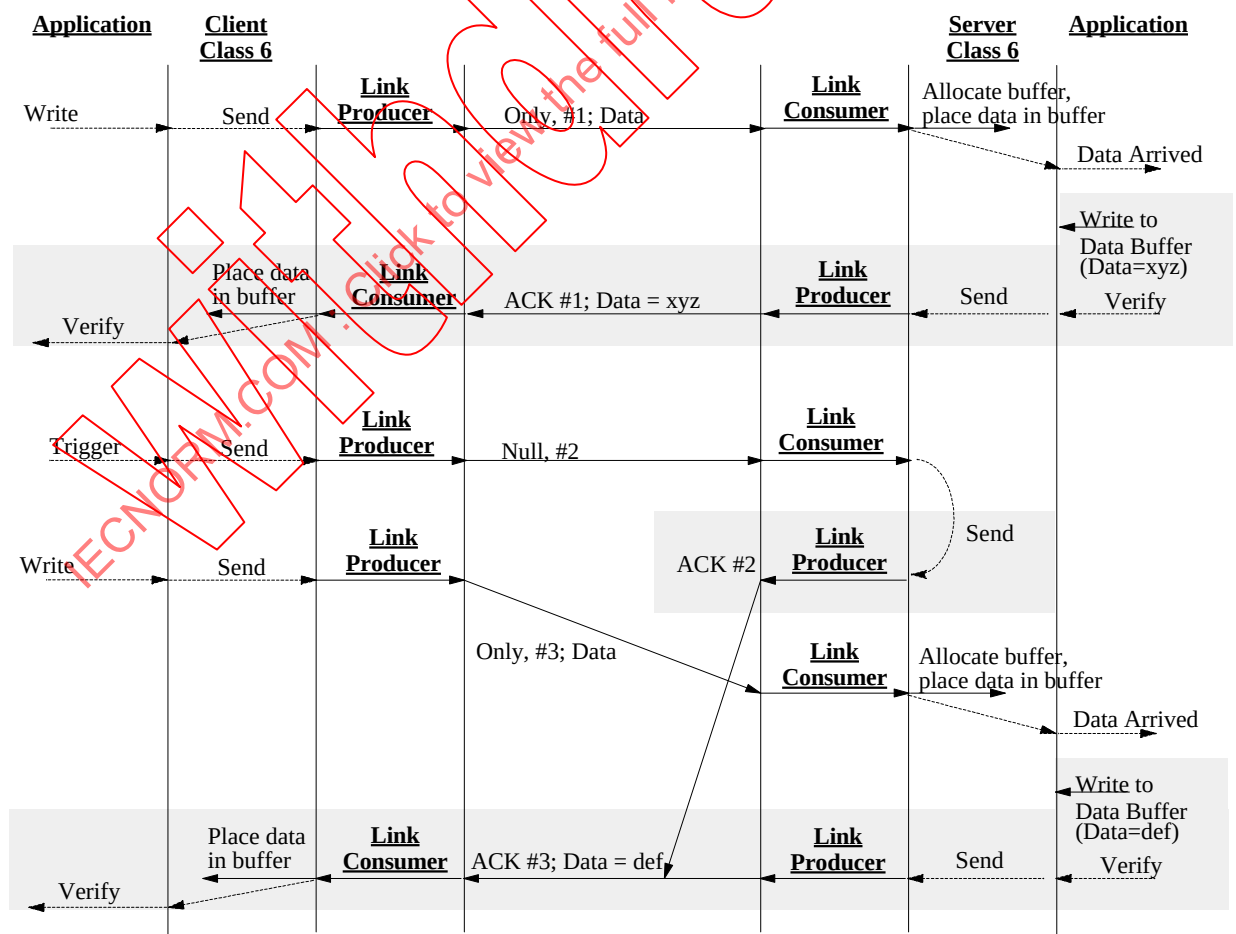


Figure 104: Sequence diagram of response overwriting ACK of null

5.6.2.12.4.6 Typical Three Fragment Sequence (Informative)

Figure 105 shows a typical sequence where the application data is too large to fit into one T-PDU. In the example shown, the application data is split into three fragments. In general, there could be many *Middle* T-PDUs. Each *Middle* T-PDU increments the Sequence Number in the Sender Header. The *Last* T-PDU shall also increment the Sequence Number.

At the receiver, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

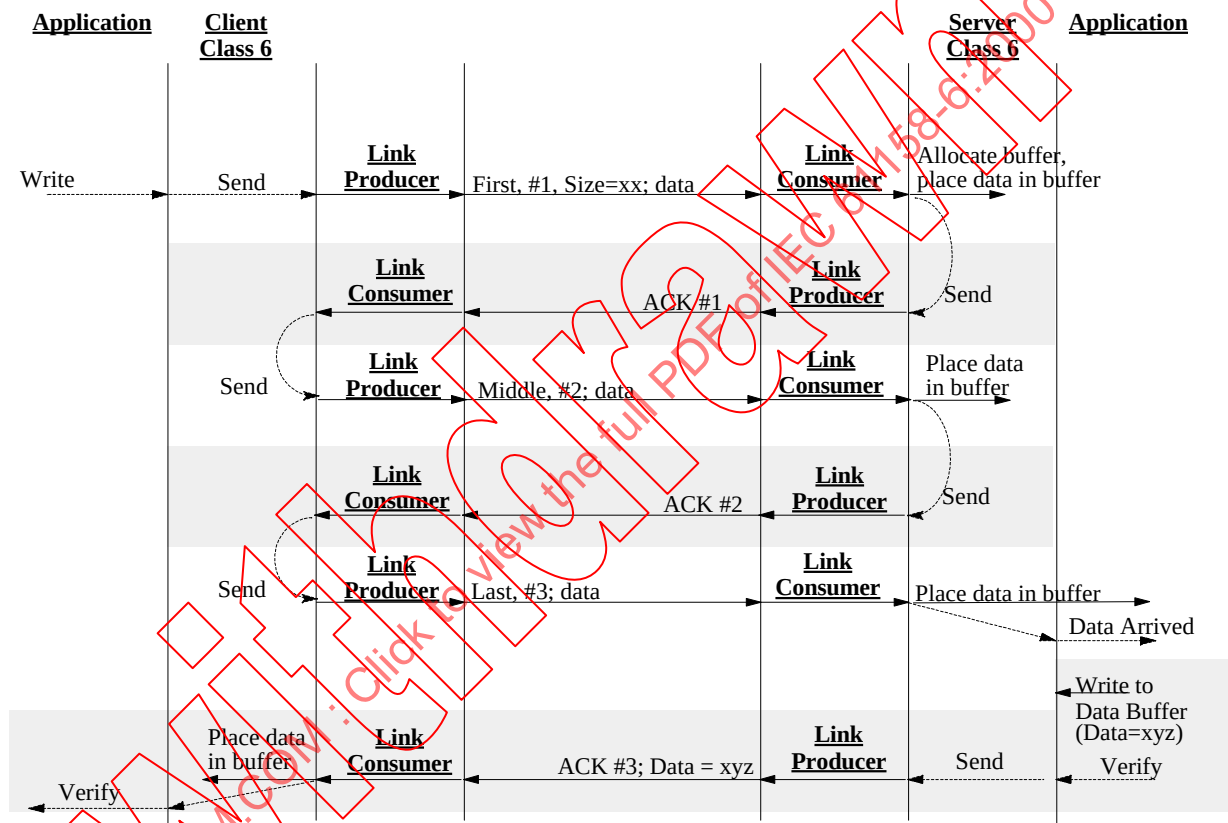


Figure 105: Sequence Diagram of Three Fragments Using Transport Class 6

5.6.2.12.4.7 Typical Fragmented Sequence With Retries (Informative)

Figure 106 shows the processing when a transmission error causes something to be lost. In this example, two transmissions are lost, one in each direction. The processing is very similar to Class 5.

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

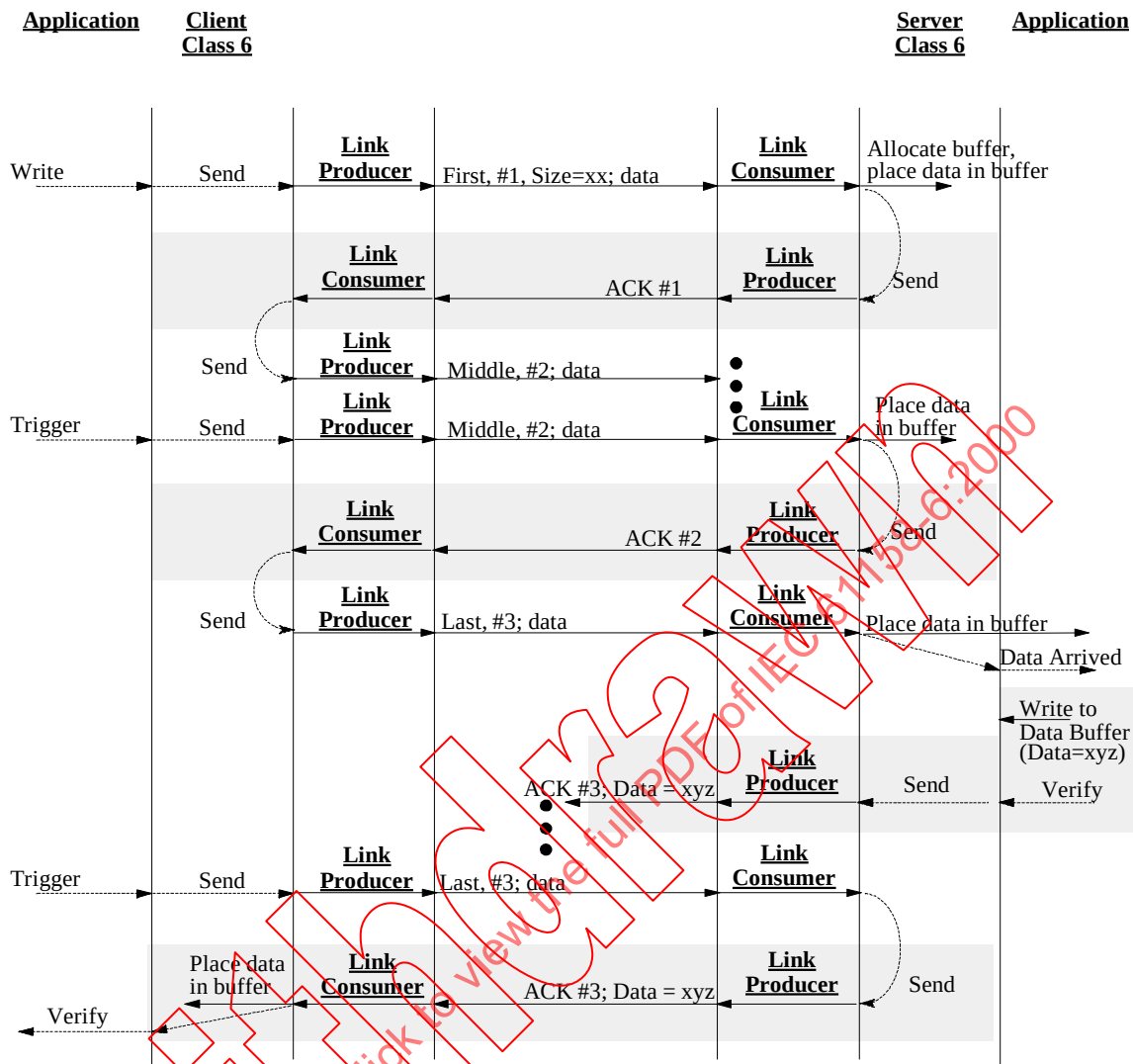


Figure 106: Sequence Diagram of Fragmentation with Retries Using Transport Class 6

5.6.2.12.4.8 Typical Two Fragment Sequence (Informative)

Figure 107 shows a special case, where there are only two fragments. In this case, there is no *Middle* T-PDU. Again, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

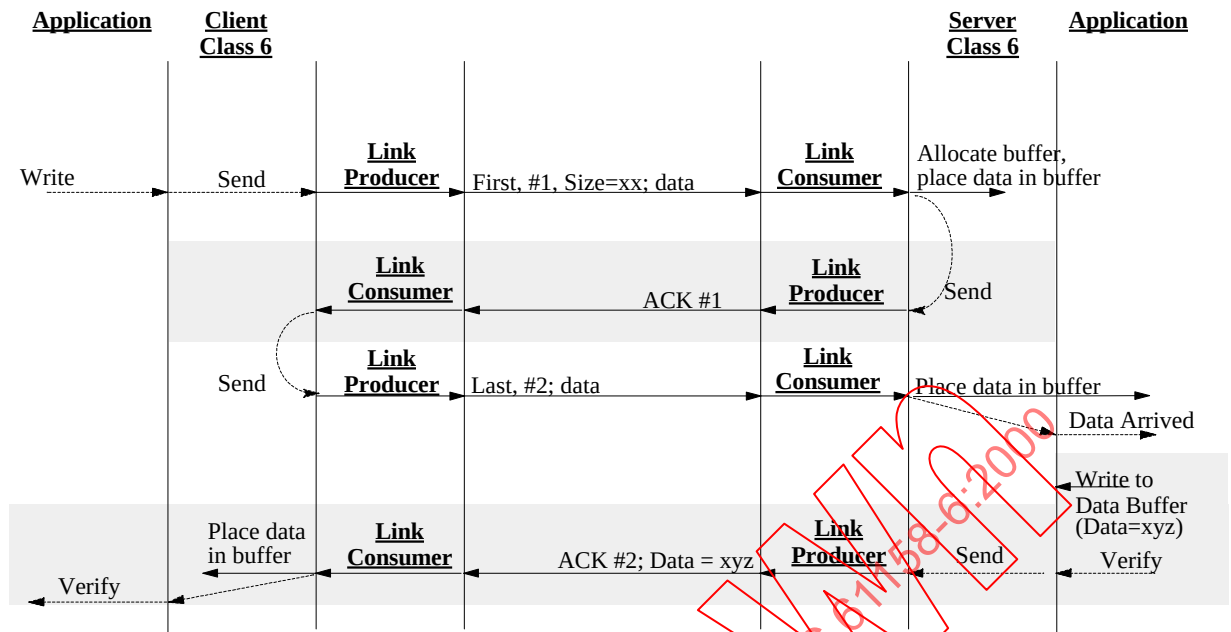


Figure 107: Sequence diagram of two fragments using transport class 6

5.6.2.12.4.9 Typical Aborted Fragmented Sequence (Informative)

Figure 108 shows a receiver recovering when the Client has been reset in the middle of a fragmented sequence. When the Server gets a *First* or *Only*, it assumes that any message which did not receive a *Last* has been aborted.

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

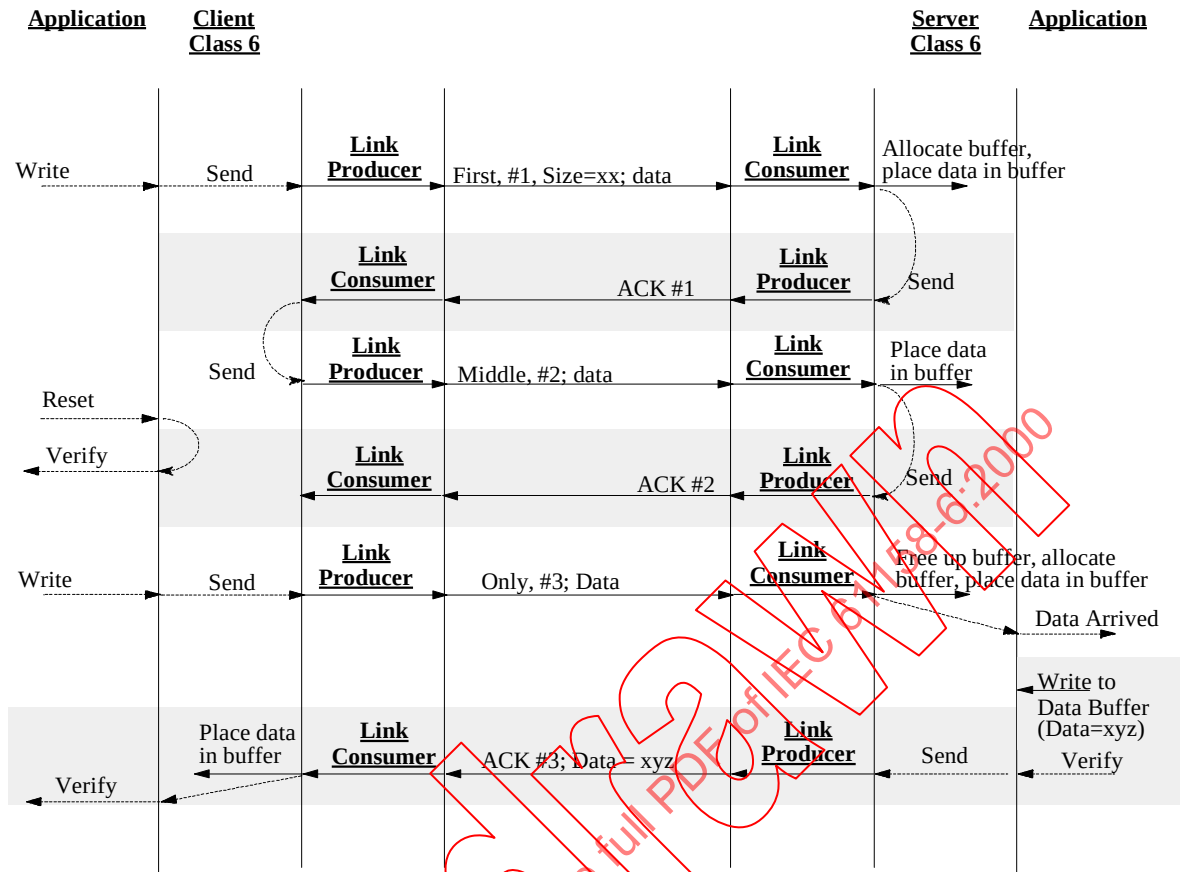


Figure 108: Sequence diagram of aborted fragmented sequence using transport class 6

5.6.2.12.5 Transport Class 6 Client

5.6.2.12.5.1 Functions

The class 6 Client transport is responsible for:

- Accepting the application's *write* that it has written a data packet to the transmit data buffer;
- Writing the application data and the transport headers to the Client T-PDU transmit buffer for each packet that the application writes to it;
- Initialising the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- Triggering the producing node of the network connection to produce the T-PDU packet on the link and send it to the consuming node of the network connection;
- Accumulating and tracking ACKs from the return connections;
- Performing retries if necessary when *triggered*;
- Notifying the application that the application data has or has not been sent.

The following internal storage is assumed:

- Current sequence number (SN);
- Try counter;
- ACK accumulator (N server nodes).

5.6.2.12.5.2 States

The defined states and their descriptions of the Class 6 Transport Client are listed in Table 327 below :

Table 327: Class 6 Transport Client States

State	Description
Ready	The instance of Transport Class 6 has been created and started.
Waiting	The instance of Transport Class 6 has been created and started. Application data has been written and the client is waiting for response.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

5.6.2.12.5.3 State Transitions

State transitions of the class 6 client transport is shown in Figure 109.

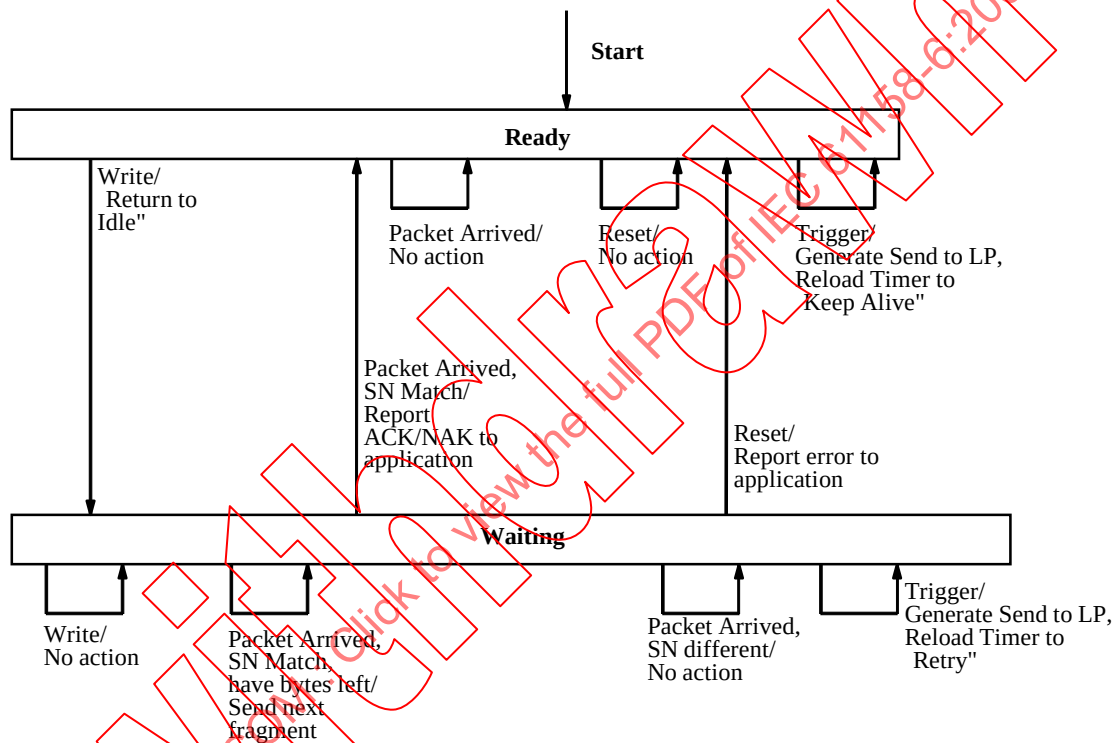


Figure 109: Class 6 client STD

5.6.2.12.5.4 State Event Matrix

State transitions of the class 6 client transport is shown in Table 328.

Table 328: Class 6 client State Event Matrix

Event	Command	Sequence Number	State	
			Ready	Waiting
Reset	n/a	n/a	No action.	Do "Return to Ready processing" with "Reset" error status.
Write	n/a	n/a	Increment Sequence number. Test if the message fits in one fragment; if it does: Set CMD to <i>Only</i> Copy message to T-PDU buffer else: Set CMD to <i>First</i> Calculate (total size / 256 bytes) - 1; put into header Copy first fragment to T-PDU buffer Set Try counter Load timer with time-out value Generate Send event to Link Producer Go to Waiting state	No action.
NewData	ACK	Different	(Not expecting anything). No action.	(Assume the other transport is ACKing something old) No action.
		Same		If the whole application message has been sent, then do "Return to Ready processing" with Success status. else, if the remainder fits in one fragment, then place the remainder in the Link Producer buffer, increment the sequence count, set the CMD to <i>Last</i>, and generate the Send event to the Link Producer else, place the next fragment in the Link Producer buffer, increment the sequence count, set the CMD to <i>Middle</i>, and generate the Send event to the Link Producer and stay in Waiting state.
	NAK	Different		(Assume the other transport is NAKing something old) No action.
		Same		Do "return to Ready processing" with NAK error status
	Other	Don't care		(Assume other end is confused.) No action.
Trigger (Retry or Keep alive timer)	Don't care	Don't care	Generate Send event to Link Producer (which re-sends the <i>Null</i> message to keep connection alive), reload timer with "keep alive" value.	Decrement Try counter, if zero: Do "return to Ready processing" with "Can't send" error status. else: Generate Send event to Link Producer (which re-sends the previous message), reload timer with retry value.

The “Return to Ready Processing” includes the following:

- Free the application data buffer;
- Generate *Verify* event to application with appropriate status;
- Increment Sequence number;
- Set CMD to *Null*;
- Generate the Send event to the Link Producer and reload the timer with the Keep Alive value;
- Go to Ready state.

5.6.2.12.6 Transport Class 6 Server

5.6.2.12.6.1 Functions

In the link to application direction, a Class 6 Server transport is responsible for:

- Accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side T-PDU receive buffer;
- Locating and reading the transport headers of each packet that the consuming node of the network connection writes to the server-side T-PDU receive buffer;
- Performing the actions specified in the state event matrix for each packet which arrives;
- Notifying the server application when it places new data into the received data buffer.

The Server Bi-directional Point-to-Point Non-Blocking transport has four states:

- Initial;
- Ready;
- Assembling;
- Waiting (for application verification);
- Previous Sequence Number (SN);
- Previous Command (CMD);

5.6.2.12.6.2 States

The defined states and their descriptions of the Class 6 Transport Server are listed in Table 329 below :

Table 329: Class 6 Transport Server States

State	Description
Initial	The instance of Transport Class 6 has been created and started.
Assembling	The instance of Transport Class 6 has been created and started. First packet of segmented application data has arrived.
Waiting	The instance of Transport Class 6 has been created and started. Subsequent packet(s) of segmented application data has(ve) arrived.
Ready	The instance of Transport Class 6 has been created and started. Last packet of segmented application data has arrived.

NOTE These four states are sub-states of the Running state described as the common feature for classes 4 to 6.

5.6.2.12.6.3 State Transitions

State transitions of the class 6 server transport are shown in Figure 110.

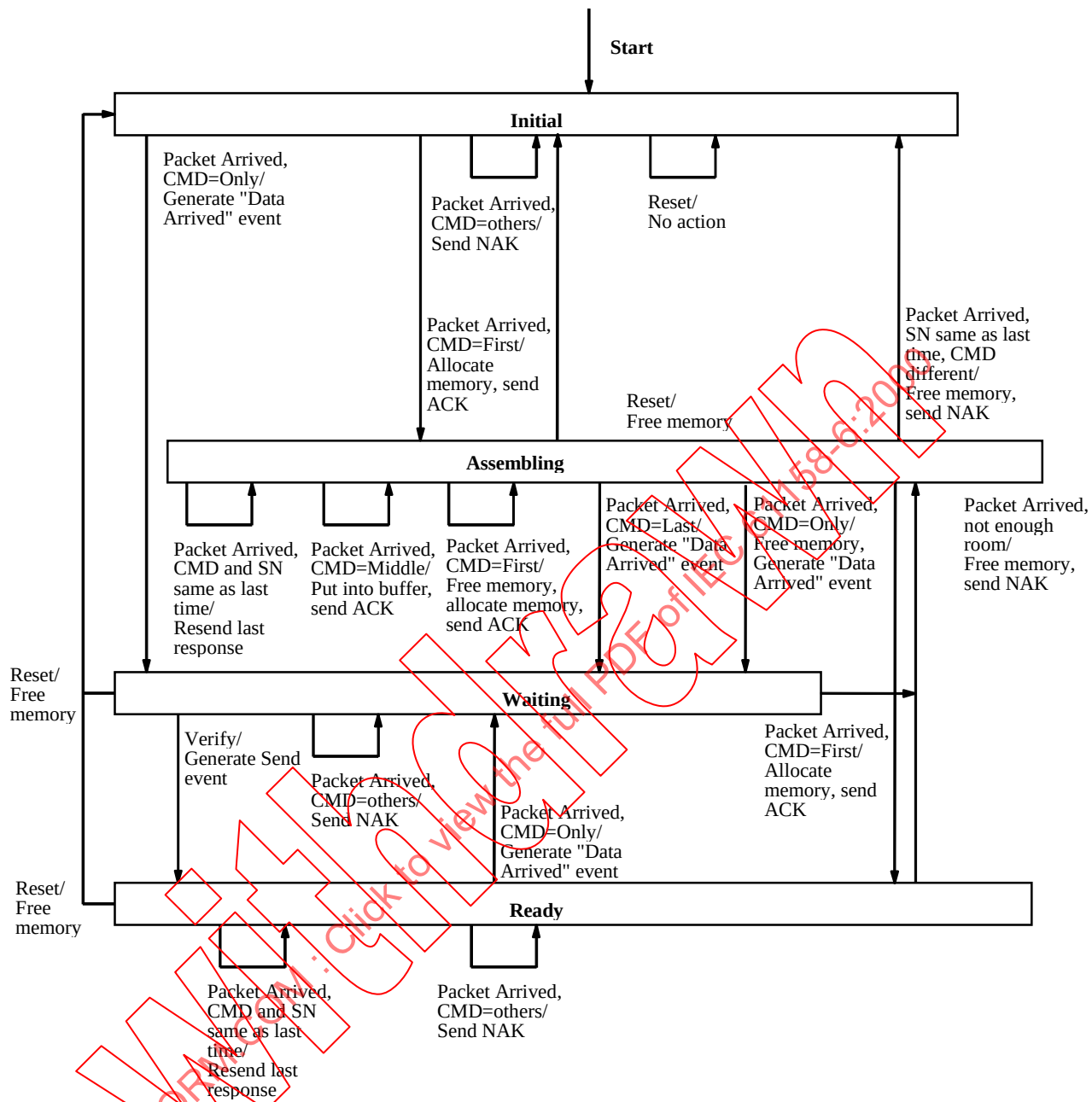


Figure 110: Class 6 server STD

The class 6 server SEM is shown in

Table 330.

The following events can happen.

- *Reset Event*;
- *Verify Event*;
- *Packet Arrived Event*, CMD= *Null*;
- *Packet Arrived Event*, CMD= *Only*
- *Packet Arrived Event*, CMD= *First*;
- *Packet Arrived Event*, CMD= *Middle*;
- *Packet Arrived Event*, CMD= *Last*;
- *Packet Arrived Event*, CMD= *Illegal command*.

NOTE There is **no** timer event used by the Server state machine. If there is a communications problem, the Client state machine shall time out. The application shall have a timer to indicate problems with the other application.

The following internal storage is assumed:

- Previous SN;
- Previous CMD received;
- Buffer Pointer;
- Room Left.

The following general guiding principles were used to develop the particular actions and transitions:

- Act like class 3 when possible;
- A *Null* shall not change state;
- When a *First* or *Only* is received when in the *Assembling* or *Waiting* states, free up the pending message and act like the server was in the *Ready* state;
- When a sequence error is detected, free up the resources and go to the *Ready* state (or stay in the *Initial* state);
- It is important to **NOT** change the CMD when responding to a retry. It might have been NAKed the last time; it shall be NAKed on a retry also.

5.6.2.12.6.4 State Event Matrix

State transitions of the class 6 server transport are shown in

Table 330.

IECNORM.COM: Click to view the full PDF of IEC 61158-6:2000

Withd2Wn

Table 330: Class 6 server SEM

Event	State			
	Initial	Ready	Assembling	Waiting
Reset	No action.	Go to Initial.	Free memory previously allocated. Go to Initial.	Free memory previously allocated. Go to Initial.
Verify	Ignore.			Generate Send event, go to Ready.
Packet Arrived, CMD=Null, SN different	Copy SN to <i>From_Recv</i> , set CMD to ACK, generate Send event, go to Ready.	Copy SN to <i>From_Recv</i> , set CMD to ACK, generate Send event, stay in Ready.	Copy SN to <i>From_Recv</i> , set CMD to ACK, generate Send event, stay in Assembling.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready. (If we ACK the <i>Null</i> , there is no way to ACK the original request. If we don't ACK the <i>Null</i> , the client shall assume we died.)
Packet Arrived, CMD=Null, SN same		If previous CMD was <i>Null</i>, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Generate Send event, stay in Ready. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, stay in Ready.	If previous CMD was <i>Null</i>, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Generate Send event, stay in Assembling. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready.	
Packet Arrived, CMD=Only, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Data Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, go to Waiting. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.		Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Data Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, go to Waiting. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.	

Event	State			
	Initial	Ready	Assembling	Waiting
Packet Arrived, CMD=Only, SN same	(Since this state does not know the previous Sequence Number, perform action as if the Sequence Number were different.)	If previous CMD was <i>Only</i> , then: Generate Send event, stay in Ready. (This is a retry, assume SN and CMD are OK. Since we are in Ready, and not Waiting, the application has verified it.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready. (Note: Shall not be in this state if previous CMD was <i>Only</i> .)	If previous CMD was <i>Only</i> , then: Generate Duplicate Arrived event, stay in Waiting. (This is a retry, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready.
Packet Arrived, CMD=First, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i> , initialise Buffer Pointer and Room Left, set CMD to ACK, generate Send event, go to Assembling state. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), generate Send event, go to Ready.	Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i> , initialise Buffer Pointer and Room Left, set CMD to ACK, generate Send event, go to Assembling. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), generate Send event, go to Ready.		
Packet Arrived, CMD=First, SN same	(Since this state does not know the previous Sequence Number, perform action as if the Sequence Number were different.)	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready. Note: Shall not be in this state if previous CMD was <i>First</i> .	If previous CMD was <i>First</i> , then: Generate Send event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready. (The Client changed the CMD but not the SN.)	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready. Note: Shall not be in this state if previous CMD was <i>First</i> .
Packet Arrived, CMD=Middle, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, copy SN to <i>From_Recv</i> , update Buffer Pointer and Room Left, set CMD to ACK, generate Send event, stay in Assembling. else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), generate Send event, go to Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready.

Event	State			
	Initial	Ready	Assembling	Waiting
Packet Arrived, CMD=Middle, SN same			If previous CMD was <i>Middle</i>, then: Generate Send event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready.	
Packet Arrived, CMD=Last, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, put buffer on Received queue, generate Data Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, generate Send event, go to Ready. else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready.
Packet Arrived, CMD=Last, SN same		If previous CMD was <i>Last</i>, then: Generate Send event, stay in Ready. (This is a retry, assume SN and CMD are OK. Since we are in Ready, and not Waiting, the application has verified it.) else: Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, stay in Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready. (Note: This shall not happen: If the previous CMD is <i>Last</i>, then we shall be in the Waiting or Ready state.)	If previous CMD was <i>Last</i>, then: Generate Duplicate Arrived event, stay in Waiting. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready.
Packet Arrived, CMD=Illegal	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Ready.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Assembling.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Waiting.

5.7 DLL Mapping Protocol Machine (DMPM)

5.7.1 Introduction

The DLL Mapping Protocol Machine is common to all the AREP types.

The Type 2 network connections that interface to the Data Link layer include the link producer(s) and link consumer(s).

Data flow of the link producer and consumer on an instance of Transport connection is shown in Figure 111.

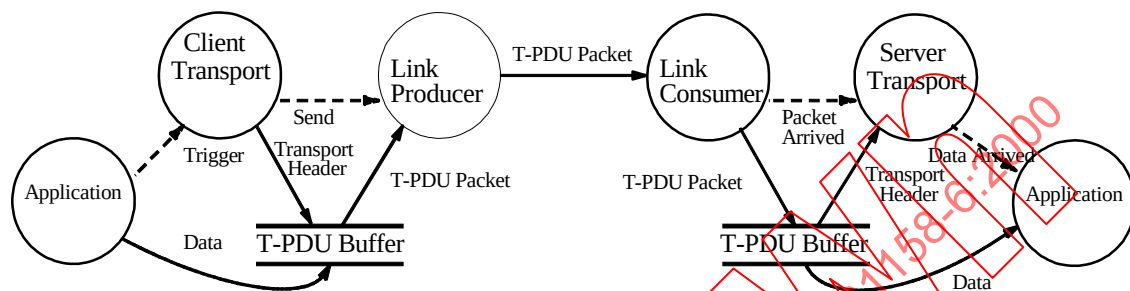


Figure 111: Data Flow Diagram for a Link Producer and Consumer

5.7.1.1 Link Producer

The link producer shall be responsible for fetching data from the T-PDU buffer and producing it on the data link after the *send* event has been received. T-PDU buffer management schemes such as overwrite, double buffered, triple buffered are not excluded, but since they are implementation specific, they are not specified here.

The link producer shall read the T-PDU packet from the T-PDU buffer and transmit it unchanged. While the T-PDU buffer is shown as a single buffer, it does not imply that the transport header and the data are stored in consecutive locations. An implementation that stored the transport header and data in non-consecutive locations is permitted as long as the link producer can find all parts of the T-PDU packet.

The application, not the link producer, shall be responsible for ensuring that the T-PDU buffer has been initialised before the first trigger has been issued.

5.7.1.2 Link Consumer

The link consumer is responsible for receiving T-PDU packets, storing them in the T-PDU buffer and notifying the server transport class that a packet has arrived. Data buffer management schemes such as overwrite, double buffered, triple buffered are not excluded, but since they are implementation specific, they are not specified here.

The link consumer shall write the T-PDU packet unchanged to the T-PDU buffer. While the T-PDU buffer is shown as a single buffer, it does not imply that the transport header and the data are stored in consecutive locations. There are no restrictions on actual implementations. An implementation that stored the transport header and data in non-consecutive locations is permitted as long as the consumer can find both parts of the T-PDU packet.

5.7.2 Primitive Definitions

5.7.2.1 Primitives Exchanged between DMPM and ARPM

List of primitives exchanged between DMPM and ARPM is shown in Table 332 and Table 333.

Table 332: Primitives issued by ARPM to DMPM

Primitive Names	Source	Associated Parameters	Functions
Send.req	ARPM	Service Dest-id T-PDU	This primitive is used to request the DMPM to transfer the T-PDU to the Data Link. It passes the T-PDU to the DMPM as a DLSDU.

Table 333: Primitives issued by DMPM to ARPM

Primitive Names	Source	Associated Parameters	Functions
Packet_Arrived.ind	DMPM	Service Source-id T-PDU	This primitive is used to pass the T-PDU received as a Data Link Layer service data unit to a designated ARPM.

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 334 .

Table 334: Parameters used with Primitives Exchanged between ARPM and DMPM

Parameter Name	Description
T-PDU	This parameter carries the transport PDU containing the user data and transport header.
DL-SDU	This parameter carries the received Data Link SDU.

5.7.2.3 Primitives Exchanged between Data Link Layer and DMPM

The request and confirmation services used to queue Lpackets with the Data Link Layer shall be of the form:

NOTE The primitives shown in Table 335 and their parameters are defined in detail in Type 2 clauses of IEC 61158-3.

Table 335: Primitives Exchanged between Data Link Layer and DMPM

Primitive Names	Source	Associated Parameters
DL_generic_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) generic_TAG
DL_generic_Tag.cnf	Data Link layer	Request_DLS_user_id (handle) TX_status
DL_generic_Tag.ind	Data Link layer	Request_DLS_user_data [] generic_TAG
DL_fixed_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) fixed_TAG destination-DLE-ID
DL_fixed_Tag.cnf	Data Link layer	Request_DLS_user_id (handle) TX_status
DL_fixed_Tag.ind	Data Link layer	user_data [] fixed_TAG source-DLE-ID
DL_tone.ind	Data Link layer	DLS_cycle

5.7.2.4 Parameters of DMPM/Data Link Layer Primitives

The parameters used with the primitives exchanged between the DMPM and the Data Link Layer are identical to those defined in the Type 2 clauses of Data Link Layer Service Definition (IEC 61158-3).

The parameters are described in Table 336 .

Table 336: Parameters used with Primitives Exchanged between DMPM and Data Link

Parameter Name	Description
Request_DLS_user_id	Provides a local means of pairing the resulting confirm primitive with the causative request primitive.
User_data	Data to be transmitted between DLS-users without alteration by the DLS-provider.
QoS	Selects one of the following priorities : scheduled high low
Generic_TAG	Connection identification or service point address identifying the remote DLSAP(s) to which the DLS is to be provided.
Fixed_TAG	Specifies the destination service point in the station identified by the DLS- Destination-DLE-ID address.
Destination-DLE-ID	Address identifying the destination station for the message using its MAC ID address.
Source-DLE-ID	Address identifying the local station from which the fixed tag DLSDU has been sent. It is a station MAC ID address on the local link.
TX_status	Status of the requested transmission : a) "OK" message successfully sent; b) "TxAbort" sending process failed; c) "Flushed" message has been removed from the pending queue before sending.
DLS_cycle	Interval count for the Network Update Time (NUT) which has just been received within the overall cycle of scheduled access intervals.

5.7.3 DMPM State Machine

5.7.3.1 DMPM States

5.7.3.1.1 Link Producer

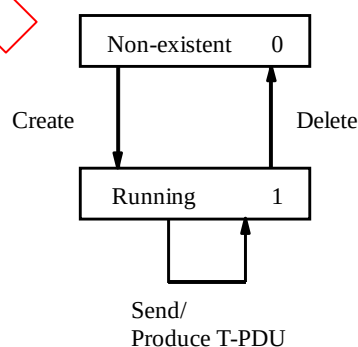
The defined states and their descriptions of the Link Producer are listed Table 337 below :

Table 337: Link Producer States

State	Description
Non-existent	The Link Producer does not exist.
Running	The Link Producer has been created and is running.

5.7.3.1.1.1 State Transition Diagram

Figure 112 shows the state transition diagram of a link producer.

**Figure 112: State Transition Diagram for a link producer**

5.7.3.1.1.2 State Event Matrix

State event matrix of the Link Producer is shown in Table 338.

Table 338: State Event Matrix of Link Producer

Event	State	
	Non-existent	Running
Create	Transition to Running	Error
Delete	Error	Transition to Non-existent
Send	Error	1) Produce T-PDU

5.7.3.1.2 Link Consumer

The defined states and their descriptions of the Link Consumer are listed in Table 339 below :

Table 339: Link Consumer States

State	Description
Non-existent	The Link Producer does not exist.
Running	The Link Producer has been created and is running.

5.7.3.1.2.1 State Transition Diagram

Figure 113 shows the state transition diagram of a link consumer.

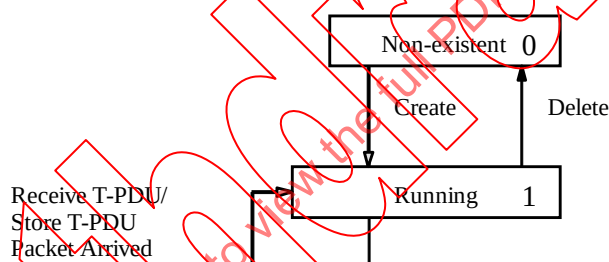


Figure 113: State Transition Diagram for a link consumer

5.7.3.1.2.2 State Event Matrix

State event matrix of the Link Producer is shown in Table 340.

Table 340 State Event Matrix of Link Consumer

Event	State	
	Non-existent	Running
Create	Transition to Running	Error
Delete	Error	Transition to Non-existent
Receive T-PDU	Error	Store T-PDU Packet Arrived

5.7.3.2 Functions used by DMPM

Type 2 fieldbus does not any specified functions.

5.7.4 Data Link Layer Service Selection

Type 2 fieldbus supports only the Data Link layer service whose primitives are noted in 5.7.2.3 above.

5.8 Protocol Options

Type 2 fieldbus does not support any protocol options.

6 Type 3

6.1 FAL Syntax Description

6.1.1 APDU Abstract Syntax

The next table below defines the abstract syntax of the DP Application Layer PDUs referred to as APDUs. The defined order of octets shall be used to convey the APDUs.

Table 341 - APDU Syntax

APDU Name	APDU Structure
DataExchange-REQ-PDU	[Outp_Data *]
DataExchange-RES-PDU	[Inp_Data*]
RD_Input-RES-PDU	[Inp_Data *]
RD_Output-RES-PDU	[Outp_Data *]
Chk_Cfg-REQ-PDU	{[Identifier_Format*], [Special_Identifier_Format*][Extended_Special_Identifier_Format*]}
Get_Cfg-RES-PDU	{[Identifier_Format*], [Special_Identifier_Format*][Extended_Special_Identifier_Format*]}
Set_Prm-REQ-PDU	Station_status, WD_Fact_1, WD_Fact_2, min_TSDR, Ident_Number, Group_Ident, [DPV1_Status_1, DPV1_Status_2, DPV1_Status_3], [User_Prm_Data*] The fields DPV1_Status_1, DPV1_Status_2 and DPV1_Status_3 shall be present if DPV1=TRUE.
Diagnosis-RES-PDU	Station_status_1, Station_status_2, Station_status_3, Diag_Master_Add, Ident_Number, {[Device_Related_Diagnosis*]^([Alarm], [Status*], [Modul_Status])}, [Identifier_Related_Diagnosis], [Channel_Related_Diagnosis*], [Revision_Number]] The field Device_Related_Diagnosis may be present if DPV1=FALSE, otherwise if DPV1=TRUE it shall not be present. In this case the fields Alarm, Status, Modul_Status may be present.
Set_Slave_Add-REQ-PDU	New_Slave_Add, Ident_Number, No_Add_Chg, [Rem_Slave_Data*]
Global_Control-REQ-PDU	Control_Command, Group_Select
Initiate-REQ-PDU	Function_Num(0x57), reserved (3 Octets), Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Para
Initiate-RES-PDU	Function_Num(0x57), Max_Len_Data_Unit, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param
Initiate-NRS-PDU	Function_Num(0xD7), Error_Decode, Error_Code_1, Error_Code_2
Abort-REQ-PDU	Function_Num(0x58), Subnet, Instance_Reason_Code
Read-REQ-PDU	Function_Num(0x5E), Slot_Number, Index, Length
Read-RES-PDU	Function_Num(0x5E), Slot_Number, Index, Length, [Data*]
Read-NRS-PDU	Function_Num(0xDE), Error_Decode, Error_Code_1, Error_Code_2
Write-REQ-PDU	Function_Num(0x5F), Slot_Number, Index, Length, [Data*]
Write-RES-PDU	Function_Num(0x5F), Slot_Number, Index, Length
Write-NRS-PDU	Function_Num(0xDF), Error_Decode, Error_Code_1, Error_Code_2
Alarm_Ack-REQ-PDU	Function_Num(0x5C), Slot_Number, Alarm_Type, Alarm_Specifier
Alarm_Ack-RES-PDU	Function_Num(0x5C), Slot_Number, Alarm_Type, Alarm_Specifier
Alarm_Ack-NRS-PDU	Function_Num(0xDC), Error_Decode, Error_Code_1, Error_Code_2
Idle-REQ-PDU	Function_Num(0x48)
Idle-RES-PDU	Function_Num(0x48)
Data_Transport-REQ-PDU	Function_Num(0x51), Slot_Number, Index, Length, [Data*]
Data_Transport-RES-PDU	Function_Num(0x51), Slot_Number, Index, Length, [Data*]
Data_Transport-NRS-PDU	Function_Num(0xD1), Error_Decode, Error_Code_1, Error_Code_2

APDU Name	APDU Structure
RM-REQ-PDU	Function_Num(0x56), Server_SAP, Send_Timeout
Get_Master_Diag-REQ-PDU	Function_Num(0x41), MDiag_Identifier
Get_Master_Diag-RES-PDU	Function_Num(0x41), MDiag_Identifier(=0..125), Diagnosis-RES-PDU Function_Num(0x41), MDiag_Identifier(=126), System_Diagnosis, Function_Num(0x41), MDiag_Identifier(=127), Master_Status Function_Num(0x41), MDiag_Identifier(=128), Data_Transfer_List
Start_Seq-REQ-PDU	Function_Num(0x42), Area_CodeUpDownload, Timeout
Start_Seq-RES-PDU	Function_Num(0x42), Area_CodeUpDownload, Timeout, Max_Len_Data_Unit
Download-REQ-PDU	Function_Num(0x43), Area_CodeUpDownload(=0..125), Add_Offset, DP_Slave_Parameter_Set Function_Num(0x43), Area_CodeUpDownload(=127), Add_Offset, Bus_Parameter_Set Function_Num(0x43), Area_CodeUpDownload(=129), Add_Offset, Statistic_Counters Function_Num(0x43), Area_CodeUpDownload(=136 to 139), Add_Offset, Master_Parameter_Set
Download-RES-PDU	Function_Num(0x43), Area_CodeUpDownload, Add_Offset
Upload-REQ-PDU	Function_Num(0x44), Area_CodeUpDownload, Add_Offset, Data_Len
Upload-RES-PDU	Function_Num(0x44), Area_CodeUpDownload(=0..125), Add_Offset, DP_Slave_Parameter_Set Function_Num(0x44), Area_CodeUpDownload(=127), Add_Offset, Bus_Parameter_Set Function_Num(0x44), Area_CodeUpDownload(=129), Add_Offset, Statistic_Counters Function_Num(0x43), Area_CodeUpDownload(=136 to 139), Add_Offset, Master_Parameter_Set
End_Seq-REQ-PDU	Function_Num(0x45)
End_Seq-RES-PDU	Function_Num(0x45)
Act_Para_Brct-REQ-PDU	Function_Num(0x46), Area_CodeActBrct
Act_Param-REQ-PDU	Function_Num(0x47), Area_CodeAct, Activate
Act_Param-RES-PDU	Function_Num(0x47), Area_CodeAct, Activate
ZERO-PDU	Octet1(0)
NULL-PDU	

NOTE 1 If a APDU consists only of user data octets the distinction is made by use of different Data Link Layer service access points (see Protocol Machines).

NOTE 2 A ZERO-PDU contains one octet with all bits set to zero. This APDU is used to indicate diagnosis from a device without inputs.

NOTE 3 A NULL-PDU contains no octets. It is used to submit no DLSDU to data link layer.

Table 342 - Substitutions

Substitution Name	Structure
Identifier_Format	Cfg_Identifier
Special_Identifier_Format	Special_Cfg_Identifier, [Length_Byte], [Length_Byte], [Manufacturer_Specific_Data*] Length_Byte fields shall always start with fields that indicate output data if present.
Extended_Special_Identifier_Format	Special_Cfg_Identifier, [Extended_Length_Byte], [Extended_Length_Byte], [Data_Type*], [Manufacturer_Specific_Data*] Extended_Length_Byte fields shall always start with fields that indicate output data if present. The field Data_Type shall only be omitted in case of an empty slot. Otherwise it shall be present in relation to the length fields.
Device_Related_Diagnosis	Header_Byte, Diagnosis_User_Data*
Identifier_Related_Diagnosis	Header_Byte, Identifier_Diagnosis_Data_Array
Channel_Related_Diagnosis	Identifier_Number, (Channel_Number, Type_of_Diagnosis)*
Alarm	Header_Byte, Alarm_Type, Slot_Number, Alarm_Specifier, [Diagnosis_User_Data*]
Status	Header_Byte, Status_Type, Slot_Number, Status_Specifier, [Diagnosis_User_Data*]
Modul_Status	Header_Byte, Status_Type(=Modul_Status), Slot_Number(=0), Status_Specifier(=0), Modul_Status_Array
Features_Supported	Features_Supported_1, Features_Supported_2
Profile_Features_Supported	Profile_Features_Supported_1, Profile_Features_Supported_1
Add_Addr_Para	S_Type, S_Len, D_Type, D_Len, S_API, S_SCL, [S_Network_Address], [S_MAC_Address], D_API, D_SCL, [D_Network_Address], [D_MAC_Address]
Master_Status	USIF_State, Ident_Number, Hardware_Release_DP, Firmware_Release_DP, Hardware_Release_User, Firmware_Release_User, reserved (9 Octets)
DP_Slave_Parameter_Set	Slave_Para_Len, SI_Flag, Slave_Type, Max_Diag_Data_Len, Max_Alarm_Len, Max_Channel_Data_Length, Diag_Upd_Delay, Alarm_Mode, Add_SI_Flag, reserved (6 Octets), Prm_Data_Len, Prm_Data, Cfg_Data_Len, Cfg_Data, Add_Tab_Len, Add_Tab, Slave_User_Data_Len, Slave_User_Data
Bus_Parameter_Set	Bus_Para_Len, DL_Add, Baud_rate, T _{SL} , min T _{SDR} , max T _{SDR} , T _{QUI} , T _{SET} , T _{TR} , G, HSA, max_retry_limit, Bp_Flag, Min_Slave_Interval, Poll_Timeout, Data_Control_Time, Alarm_Max, Max_User_Global_Control, reserved (4 Octets), Master_User_Data_Len, Master_Class2_Name, Master_User_Data*
Master_Parameter_Set	Bus_Parameter_Set, DP_Slave_Parameter_Set*
Statistic_Counters	Counter_Group*
Counter_Group	if Add_Offset 0..126: Frame_sent_count(Add_Offset), Error_count(Add_Offset) if Add_Offset 127: SD_sent_count, SD_error_count
Add_Tab	[Number_of_Entries], [Add_Tab_Entry_Header, I/O_Data_Length, IO_Config_Address, Host_Address]*

NOTE The DP_Slave_Parameter_Set and the Bus_Parameter_Set may exceed the maximum APDU size to a size up to 64 Kbytes. The conveyance of the data is therefore segmented by means of a window addressed with the parameter Add_Offset.

6.1.2 Data Types

Refer to FAL Type 1 Data Type Description in subclause 4.1.4.

6.2 Transfer Syntax

6.2.1 Coding of Basic Data Types

There is no special coding of basic data types for Type 3. The Traditional Encoding Rules (TER) of Type 1 are used to encode the values of basic data types (e.g. Boolean) (see subclause 4.2.1.2.1.2.4).

NOTE The encoding of the values are used for the contents of user data only. Type 3 do not use TER to encode the entire APDU in conjunction with ASN1. Identification information and tagging is not present.

Floating Point:

For type 3 coding Floating Point means Floating Point32.

6.2.2 Coding Section related to Data Exchange PDUs

6.2.2.1 Coding of the Field Outp_Data

One of the following data types shall be used for each Outp_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.2.2 Coding of the Field Inp_Data

One of the following data types shall be used for each Inp_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

NOTE Data types with variable length (Visible String, Octet String, Time of Day, Time Difference) for Inp_Data or Outp_Data are only once present in the DataExchange-RES-PDU or DataExchange-REQ-PDU. The user data correspond to the contents of the Configuration-PDU.

6.2.3 Coding Section related to Slave Diagnosis PDUs

6.2.3.1 Coding of the Field Station_status_1

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Diag.Station_Non_Existent

Shall be set to zero in case of Diagnoses-RES-PDU

Value (0): means FALSE if Get_Master_Diag-RES-PDU

Value (1): means TRUE if Get_Master_Diag-RES-PDU

Bit 1: Diag.Station_Not_Ready

FALSE (0), TRUE (1)

Bit 2: Diag.Cfg_Fault (Configuration Fault)

FALSE (0), TRUE (1)

Bit 3: Diag.Ext_Diag (Extended Diagnosis)

FALSE (0), TRUE (1)

Bit 4: Diag.Not_Supported

FALSE (0), TRUE (1)

Bit 5: Diag.Invalid_Slave_Response

Shall be set to zero if Diagnoses-RES-PDU

Value (0): means FALSE if Get_Master_Diag-RES-PDU

Value (1): means TRUE if Get_Master_Diag-RES-PDU

Bit 6: Diag.Prm_Fault (Parameter Fault)

FALSE (0), TRUE (1)

Bit 7: Diag.Master_Lock

Shall be set to zero if Diagnoses-RES-PDU

Value (0): means FALSE if Get_Master_Diag-RES-PDU

Value (1): means TRUE if Get_Master_Diag-RES-PDU

6.2.3.2 Coding of the Field Station_status_2

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Diag.Prm_Req (Parameterisation requested)

FALSE (0), TRUE (1)

Bit 1: Diag.Stat_Diag (Static Diagnosis)

FALSE (0), TRUE (1)

Bit 2: DP (DP Protocol)

Shall always set to one

Bit 3: Diag.WD_On (Watchdog on)

FALSE (0), TRUE (1)

Bit 4: Diag.Freeze_Mode

FALSE (0), TRUE (1)

Bit 5: Diag.Sync_Mode (Synchronisation Mode)

FALSE (0), TRUE (1)

Bit 6: reserved

Bit 7: Diag.Deactivated

Shall be set to zero if Diagnoses-RES-PDU

Value (0): means FALSE if Get_Master_Diag-RES-PDU

Value (1): means TRUE if Get_Master_Diag-RES-PDU

6.2.3.3 Coding of the Field Station_status_3

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 6: reserved

Bit 7: Diag.Ext_Diag_Overflow (Overflow of extended Diagnosis)

FALSE (0), TRUE (1)

6.2.3.4 Coding of the Field Diag_Master_Add

This field shall be coded as data type Unsigned8 with the following values:

Decimal (0-125): means the address of the DP-Master (Class 1) which has parameterised this DP-Slave

Decimal (255): means DP-Slave has not been parameterised or has got invalid parameterisation.

Decimal (126-254): not allowed

6.2.3.5 Coding of the Field Ident_Number

This field shall be coded as data type Unsigned16.

6.2.3.6 Coding of the Field Header_Byte

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 5: Block_Length

It shall contain the length of the diagnosis block. The Header_Byte itself shall always be counted.

For the base diagnosis format of the Device Related Diagnosis and for the Identifier Related Diagnosis the values shall be in the range of decimal 2 to 63. For the DPV1 diagnosis format for the Device Related Diagnosis the values shall be in the range of decimal 4 to 63.

Bit 6 to 7: Selection

Decimal (0): means Device Related Diagnosis

Decimal (1): means Identifier Related Diagnosis

6.2.3.7 Coding of the Field Alarm_Type

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 6: Alarm_Type

The Alarm_Type shall contain the values according to the table below:

Table 343 - Alarm_Type Range

Value (decimal)	Meaning
0	Shall not be used
1	Diagnostic_Alarm
2	Process_Alarm
3	Pull_Alarm
4	Plug_Alarm
5	Status_Alarm
6	Update_Alarm
7 - 31	Shall not be used
32 - 126	manufacturer-specific
127	Shall not be used

Bit 7: shall be zero (identifying Alarm)

6.2.3.8 Coding of the Field Status_Type

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 6: Status_Type

The Status_Type shall contain values according to the table below.

Table 344 - Status_Type Value Range

Value (decimal)	Meaning
0	Shall not be used
1	Status_Message
2	Modul_Status
3 to 31	Shall not be used
32 to 126	manufacturer-specific
127	Shall not be used

Bit 7: shall be one (identifying Status)

6.2.3.9 Coding of the Field Slot_Number

This field shall be coded as data type Unsigned8. The value shall be taken from the range zero to 254. The value 255 is reserved and shall not be used.

6.2.3.10 Coding of the Field Alarm_Specifier

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 1: Alarm_Specifier

This bit group shall contain the alarm specifier with the values according to the table below:

Table 345 - Alarm_Specifier

value (decimal)	meaning	Comment
0	no further differentiation	
1	Error appears and Slot disturbed	the slot generates an alarm due to an error
2	Error disappears and Slot is okay	the slot generates an alarm and indicates that the slot has no further errors
3	Error disappears and Slot is still disturbed	the slot generates an alarm and indicates that the slot has still further errors

Bit 2: Additional_Acknowledge

Value (0): means no additional acknowledge used

Value (1): means additional acknowledge is used

Bit 3 to 7: Sequence_Number

The value range shall be from zero to 31 (decimal).

6.2.3.11 Coding of the Field Status_Specifier

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 1: Status_Specifier

Decimal (0): means no further differentiation

Decimal (1): means status appears

Decimal (2): means status disappears

Decimal (3): reserved, shall not be used

Bit 2 to 7: reserved

6.2.3.12 Coding of the Field Diagnosis_User_Data

One of the following data types shall be used for each Diagnosis_User_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.3.13 Coding of the Field Modul_Status_Array

The field Modul_Status_Array is an array of octets that shall contain at least one and at most 61 octets referred to as Modul_Status_Octet. A Modul_Status_Octet shall consist of at least one and at most 4 module stati referred to as Modul_Status_Entry_1 to Modul_Status_Entry_4 coded in two bits each. Therefore, the number of Modul_Status_Octet corresponds to the number of configured modules within a device and shall be calculated as follows

a) $N = 1$ if $M_{\text{highest}} \leq 4$

b) $N = M_{\text{highest}} \text{ DIV } 4 + 1$ if $M_{\text{highest}} \text{ MOD } 4 \neq 0$

c) $N = M_{\text{highest}} \text{ DIV } 4$ if $M_{\text{highest}} \text{ MOD } 4 = 0$

where

N is the number of octets `Modul_Status_Octet` or the number of array elements,

M_{highest} is the highest number of configured modules within a device (maximum 244).

The last `Modul_Status_Octet` (octet N) may not contain all 4 module status entries. If $M_{\text{highest}} \text{ MOD } 4 \neq 0$ the octet N is not fully filled and the remaining bits shall be set to zero referred to as Padding Bits.

The stati of the device's modules shall be structured in ascending order without gaps. The status of module number one is always placed in `Modul_Status_Entry_1` of the `Modul_Status_Octet` number one. The coding of a `Modul_Status_Octet` shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

NOTE The term "configured modules" addresses physical or virtual modules of a device that have had a related format field in the `Chk_Cfg-REQ-PDU`. Modules that are not part of the configuration are purely local and therefore not related to the `Modul_Status_Array` field.

Bit 0 to 1: Modul_Status_Entry_1

It shall contain the module status of a configured module with the number that meets $m \text{ MOD } 4 = 1$. Padding bits shall not be used here.

Bit 2 to 3: Modul_Status_Entry_2

It shall contain the module status of a configured module with the number that meets $m \text{ MOD } 4 = 2$. Otherwise padding bits (00 binary) shall be used.

Bit 4 to 5: Modul_Status_Entry_3

It shall contain the module status of a configured module with the number that meets $m \text{ MOD } 4 = 3$. Otherwise padding bits (00 binary) shall be used.

Bit 6 to 7: Modul_Status_Entry_4

These 2 bits shall contain the module status of a configured module with the number that meets $m \text{ MOD } 4 = 0$. Otherwise padding bits (00 binary) shall be used.

where

m is the number of the current module with $1 \leq m \leq N$.

The figure below illustrates the arrangement of `Modul_Status_Entry` (1 to 4) fields as an example for a device with 6 configured modules.

Bit Number		7	6	5	4	3	2	1	0
<code>Modul_Status_Octet 1</code>	status module number 4	status module number 3		status module number 2		status module number 1			
<code>Modul_Status_Octet 2</code>	Padding Bits	Padding Bits		status module number 6		status module number 5			
	<code>Modul_Status_Entry_4</code>	<code>Modul_Status_Entry_3</code>		<code>Modul_Status_Entry_2</code>		<code>Modul_Status_Entry_1</code>			

Figure 114 - Example `Modul_Status_Array`

Each entry `Modul_Status_Entry_1` to `Modul_Status_Entry_4` shall contain the module status according to the table below:

Table 346 - Range of `Modul_Status_Entry` (1-4)

value (decimal)	Meaning
0	data valid
1	data invalid: the data of the corresponding module are not valid due to an error (e.g. short circuit)
2	data invalid/wrong module: the data of the corresponding module are not valid, due to a wrong module in place
3	data invalid/no module: the data of the corresponding module are not valid, because there is no module in place

6.2.3.14 Coding of the Field Identifier_Diagnosis_Data_Array

The field Identifier_Diagnosis_Data_Array is an array of octets that shall contain at least one and at most 31 octets referred to as Identifier_Diagnosis_Data_Octet. A Identifier_Diagnosis_Data_Octet shall consist of at least one and at most 8 diagnosis entries referred to as Identifier_Diagnosis_Entry_1 to Identifier_Diagnosis_Entry_8. Therefore, the number of Identifier_Diagnosis_Data_Octet corresponds to the number of configured modules within a device and shall be calculated as follows

- a) $N = 1$ if $M_{\text{highest}} \leq 8$
- b) $N = M_{\text{highest}} \text{ DIV } 8 + 1$ if $M_{\text{highest}} \text{ MOD } 4 \neq 0$
- c) $N = M_{\text{highest}} \text{ DIV } 8$ if $M_{\text{highest}} \text{ MOD } 4 = 0$

where

N is the number of octets Identifier_Diagnosis_Data_Octet or the number of array elements,

M_{highest} is the highest number of configured modules within a device (maximum 244).

The last Identifier_Diagnosis_Data_Octet (octet N) may not contain all 8 diagnosis entries. If $M_{\text{highest}} \text{ MOD } 8 \neq 0$ 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 identifier diagnosis of the device's modules shall be structured in ascending order without gaps. The identifier diagnosis of module number one is always placed in Identifier_Diagnosis_Entry_1 of the Identifier_Diagnosis_Data_Octet number one. The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Identifier_Diagnosis_Entry_1

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 1$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall not be used here.

Bit 1: Identifier_Diagnosis_Entry_2

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 2$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

Bit 2: Identifier_Diagnosis_Entry_3

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 3$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

Bit 3: Identifier_Diagnosis_Entry_4

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 4$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

Bit 4: Identifier_Diagnosis_Entry_5

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 5$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

Bit 5: Identifier_Diagnosis_Entry_6

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 6$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

Bit 6: Identifier_Diagnosis_Entry_7

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 7$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

Bit 7: Identifier_Diagnosis_Entry_8

This bit shall be set to one if the module with the number that meets $m \text{ MOD } 8 = 0$ indicates diagnosis otherwise it shall be set to zero. A padding bit shall be used if there is no module configured.

where

m is the number of the current module with $1 \leq m \leq N$.

6.2.3.15 Coding of the Field Identifier_Number

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 5: Identifier_Number

The values zero to 63 (decimal) are valid.

Bit 6: shall be zero

Bit 7: shall be one

6.2.3.16 Coding of the Field Channel_Number

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 5: Channel_Number

The values zero to 63 (decimal) are valid.

Bit 6 to 7: Input_Output_Selection

Decimal (0): reserved, shall not be used

Decimal (1): means input

Decimal (2): means output

Decimal (3): means input and output

6.2.3.17 Coding of the Field Type_of_Diagnosis

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 4: Error_Type

The Error_Type shall be set as shown in the table below.

Table 347 - Error type

Value (decimal)	Meaning
0	Shall not be used
1	short circuit
2	Undervoltage
3	Overvoltage
4	Overload
5	Overtemperature
6	line break
7	upper limit value exceeded
8	lower limit value exceeded
9	Error
10	Shall not be used
.	
15	Shall not be used
16	manufacturer specific
.	
31	manufacturer specific

Bit 5 to 7: Channel_Type

Decimal (0): unspecific, may be used for any type

Decimal (1): means bit

Decimal (2): means 2 bit

Decimal (3): means 4 bit

Decimal (4): means byte

Decimal (5): means word

Decimal (6): means 2 words

Decimal (7): reserved, shall not be used

6.2.3.18 Coding of the Field Revision_Number

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 5: Revision_Number

The values one to 63 (decimal) are valid.

Bit 6: shall be set to one**Bit 7: shall be set to one****6.2.4 Coding Section related to Parameterisation PDU****6.2.4.1 Coding of the Field Station_status**

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: reserved**Bit 1: reserved****Bit 2: reserved****Bit 3: WD_On (Watchdog on)**

FALSE (0), TRUE (1)

Bit 4: Freeze_Req

Value(0): Freeze mode not requested

Value(1): Freeze mode requested

Bit 5: Sync_Req

Value(0): Sync mode not requested

Value(1): Sync mode requested

Bit 7: Lock_Req

Shall be used as described in the table below.

Bit 6: Unlock_Req

Shall be used as described in the table below.

Table 348 - Specification of the Bits Lock_Req and Unlock_Req

Bit 7	Bit 6	Meaning
0	0	The parameter min T _{SDR} can be changed. All other parameters remain unchanged.
0	1	The DP-Slave will be unlocked for other Masters.
1	0	The DP-Slave is locked for other Masters, all parameters are accepted (exception: min T _{SDR} = 0)
1	1	The DP-Slave is unlocked for other Masters.

6.2.4.2 Coding of the Field WD_Fact_1

This field shall be coded as data type Unsigned8 with the following value range:

Decimal 1 to 255

6.2.4.3 Coding of the Field WD_Fact_2

This field shall be coded as data type Unsigned8 with the following value range:

Decimal 1 to 255

6.2.4.4 Coding of the Field min_TSDR

This field shall be coded as data type Unsigned8 with the following value range:

Decimal(0 to max T_{SDR} (see Part 4)) if DP operation

Decimal(0): means current value remains unchanged

6.2.4.5 Coding of the Field Group_Ident

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Group_1

Bit 1: Group_2

Bit 2: Group_3

Bit 3: Group_4

Bit 4: Group_5

Bit 5: Group_6

Bit 6: Group_7

Bit 7: Group_8

Each of the bits above shall set to one if the device belongs to the indicated group number. Otherwise it shall be set to zero.

NOTE A device may belong to no groups (Group_Ident=decimal(0)) or all groups (Group_Ident=decimal(255)).

6.2.4.6 Coding of the Field User_Prm_Data

One of the following data types shall be used for each User_Prm_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.4.7 Coding of the Field DPV1_Status_1

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: reserved

Bit 1: reserved

Bit 2: WD_Base_1ms

Value(0): Watchdog time base equals 10 milliseconds

Value(1): Watchdog time base equals 1 millisecond

Bit 3 to 5 : reserved

Bit 6: Fail_Safe

FALSE(0), TRUE(1)

Bit 7: DPV1_Enable

FALSE(0), TRUE(1)

6.2.4.8 Coding of the Field DPV1_Status_2

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Check_Cfg_Mode

Value(0): normal check (restrictive)

Value(1): user specific check (more flexible)

Bit 1: reserved

Bit 2: Enable_Update_Alarm

FALSE(0), TRUE(1)

Bit 3: Enable_Status_Alarm

FALSE(0), TRUE(1)

Bit 4: Enable_Manufacturer_Specific_Alarm

FALSE(0), TRUE(1)

Bit 5: Enable_Diagnostic_Alarm

FALSE(0), TRUE(1)

Bit 6: Enable_Process_Alarm

FALSE(0), TRUE(1)

Bit 7: Enable_Pull_Plug_Alarm

FALSE(0), TRUE(1)

6.2.4.9 Coding of the Field DPV1_Status_3

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 2: Alarm_Mode

Decimal(0): 1 alarm of each type

Decimal(1): 2 alarms in total

Decimal(2): 4 alarms in total

Decimal(3): 8 alarms in total

Decimal(4): 12 alarms in total

Decimal(5): 16 alarms in total

Decimal(6): 24 alarms in total

Decimal(7): 32 alarms in total

Bit 3 to 7: reserved

6.2.5 Coding Section related to Configuration PDUs

6.2.5.1 Coding of the Field Cfg_Identifier

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 3: Data_Length

Decimal(0): means 1 byte or 1 word according to the Length_Format

Decimal(1): means 2 bytes or 2 words according to the Length_Format

Decimal(2): means 3 bytes or 3 words according to the Length_Format

Decimal(3): means 4 bytes or 4 words according to the Length_Format

Decimal(4): means 5 bytes or 5 words according to the Length_Format

...

Decimal(15): means 16 bytes or 16 words according to the Length_Format

Bit 4 to 5: Input_Output_Selection

Decimal(0): shall not be used here, means specific identifier format

Decimal(1): means input

Decimal(2): means output

Decimal(3): means input and output

Bit 6: Length_Format

Value(0) : means byte structure used

Value(1) : means word structure used (high byte transferred first, big-endian)

Bit 7: Consistency

Value(0): means byte or word consistency

Value(1): means consistency over the whole length

6.2.5.2 Coding of the Field Special_Cfg_Identifier

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 3: Length_of_Manufacturer_Specific_Data

The length information of the manufacturer specific data shall be in dependency of the use in the Chk_Cfg-REQ-PDU or Get_Cfg-RES-PDU as shown in the two table below.

Bit 4 to 5: shall be set to zero (decimal)

Bit 6 to 7: Input_Output_Selection

Decimal(0): means empty slot, no input or output data for this module

Decimal(1): means one length octet for input data follows

Decimal(2): means one length octet for output data follows

Decimal(3): means one length octet for output data followed by one length octet for input data follows

Table 349 - Range of Length_of_Manufacturer_Specific_Data if used in Chk_Cfg-REQ-PDU

Value (decimal)	Meaning
0	No manufacturer specific data follow; no data in Real_Cfg_Data.
1 to 14	Manufacturer specific data of specified length follow; these shall be identical with the data in Real_Cfg_Data.
15	No manufacturer specific data follow; the verification can be omitted

Table 350 - Range of Length_of_Manufacturer_Specific_Data if used in Get_Cfg-RES-PDU

Value (decimal)	Meaning
0	No manufacturer specific data follow
1 to 14	Manufacturer specific data with specified length follow
15	Not allowed

6.2.5.3 Coding of the Fields Length_Byte

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 5: Data_Length

Decimal(0): means 1 byte or 1 word according to the Length_Format

Decimal(1): means 2 bytes or 2 words according to the Length_Format

...

Decimal(63): means 64 bytes or 64 words according to the Length_Format

Bit 6: Length_Format

Value(0): means byte structure used

Value(1): means word structure used (high byte transferred first, big-endian)

Bit 7: Consistency

Value(0): means byte or word consistency

Value(1): means consistency over the whole length indicated in Data_Length

6.2.5.4 Coding of the Field Manufacturer_Specific_Data

One of the following data types shall be used for each Manufacturer_Specific_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.5.5 Coding of the Field Extended_Length_Byte

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 5: Data_Length

Decimal(0): means 1 byte

Decimal(1): means 2 bytes

Decimal(2): means 3 bytes

Decimal(3): means 4 bytes

Decimal(4): means 5 bytes

...

Decimal(63): means 64 bytes

Bit 6: Format

This bit shall be set to zero that means byte structure is used.

Bit 7: Consistency

This bit shall be set to one that means consistency over the whole length.

6.2.5.6 Coding of the Field Data_Type

This field shall be coded as data type Unsigned8 with values according to the code column at the table below.

Table 351 - Values (codes) for data types

Data type name	Code (decimal)	Number of octets for the data type
Boolean	1	1
Integer8	2	1
Integer16	3	2
Integer32	4	4
Unsigned8	5	1
Unsigned16	6	2
Unsigned32	7	4
Floating Point 32	8	4
Visible String	9	1, 2, 3, ...
Octet String	10	1, 2, 3, ...
Date	11	7
Time Of Day	12	4
Time Difference	13	4
Time Of Day (with Days)	14	6
Time Difference (with Days)	15	6
	16 to 31	Shall not be used
	32 to 63	User specific
	64 to 127	Shall not be used
	128 to 255	User specific (no data type coding)

Sequences of data types shall be coded as a list of simple data types. The sum of the lengths of the data types shall be corresponding the input- or output data length. Data types with variable length (visible string, octet string, time of day, time difference) shall not be handled in a sequence, but as single element.

6.2.6 Coding Section related to Global Control PDUs

6.2.6.1 Coding of the Field Control_Command

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: reserved

Bit 1: Clear_Data

Value(0): no command

Value(1): clear output (set output data in safe state)

Bit 2: UnFreeze

Value(0): no command

Value(1): cancels Freeze command

Bit 3: Freeze

Value(0): no command

Value(1): freeze input values

Bit 4: Unsync

Value(0): no command

Value(1): cancels Sync command

Bit 5: Sync

Value(0): no command

Value(1): synchronise output values

Bit 6: reserved

Bit 7: reserved

Table 352 - Specification of the bits for Un-/Sync and Un-/Freeze

Bit 2 resp. 4	Bit 3 resp. 5	Meaning
0	0	no function
0	1	function is activated
1	0	function is deactivated
1	1	function is deactivated

6.2.6.2 Coding of the Field Group_Select

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Group_1

Bit 1: Group_2

Bit 2: Group_3

Bit 3: Group_4

Bit 4: Group_5

Bit 5: Group_6

Bit 6: Group_7

Bit 7: Group_8

Each of the bits above shall set to one if the control command belongs to the indicated group number. Otherwise it shall be set to zero.

NOTE A control command may belong to no groups (Group_Ident=decimal(0)) or all groups (Group_Ident=decimal(255)).

6.2.7 Coding Section related to Function Identification and Errors

6.2.7.1 Coding of the Field Function_Num

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meanings

Bit 0 to 4: Function_Code / Error_Code

This bits contain the Function_Code or the Error_Code corresponding to the Frame_Selector. The Function_Code shall be set according to the table below.

Bit 5 to 6: PDU_Identifier

These two bits shall be set to 2 (decimal) to indicate a DP-PDU.

Bit 7: Frame_Selector

Value(0): means request frame or positive response frame

Value(1): means error or negative response frame

The field Function_Num is transferred in the request and response frame. If the service is processed correctly, bit 7 of Function_Num in the response APDU shall be set to zero. In case of an error an error frame is transferred. In this APDU Function_Num contains an error code and the bit 7 shall be set to one.

Table 353 - Coding of the Function_Code/ Function_Num

Frame_ Selector (decimal)	PDU_ Identifier (decimal)	Function_Code (decimal)	Meaning	Complete field: Function_Num (hexadecimal)
0	2	0	Shall not be used	0x40
0	2	1	Get_Master_Diag	0x41
0	2	2	Start_Seq	0x42
0	2	3	Download	0x43
0	2	4	Upload	0x44
0	2	5	End_Seq	0x45
0	2	6	Act_Para_Brct	0x46
0	2	7	Act_Param	0x47
0	2	8	Idle	0x48
0	2	9 to 16	Shall not be used	0x49 to 0x50
0	2	17	Data_Transport	0x51
0	2	18 to 21	Shall not be used	0x52 to 0x55
0	2	22	RM	0x56
0	2	23	Initiate	0x57
0	2	24	Abort	0x58
0	2	25	Shall not be used	0x59
0	2	26	Shall not be used	0x5A
0	2	27	Shall not be used	0x5B
0	2	28	Alarm_Ack	0x5C
0	2	29	Shall not be used	0x5D
0	2	30	Read	0x5E
0	2	31	Write	0x5F

The Error_Code shall be set according to the next table below:

Table 354 - Coding of the Error_Code / Function_Num

Frame_ Selector (decimal)	PDU_ Identifier (decimal)	Error_Code (decimal)	Meaning	Complete field: Function_Num (hexadecimal)
1	2	0	Shall not be used	0xC0
1	2	1	FE	0xC1
1	2	2	NI	0xC2
1	2	3	AD	0xC3
1	2	4	EA	0xC4
1	2	5	LE	0xC5
1	2	6	RE	0xC6
1	2	7	IP	0xC7
1	2	8	SC	0xC8
1	2	9	SE	0xC9
1	2	10	NE	0xCA
1	2	11	DI	0xCB
1	2	12	NC	0xCC
1	2	13	TO	0xCD
1	2	14	CA	0xCE
1	2	15 to 16	Shall not be used	0xCF to 0xD0
1	2	17	Error Data Transport	0xD1
1	2	18 to 22	Shall not be used	0xD2 to 0xD6
1	2	23	Error Initiate	0xD7
1	2	24	Shall not be used	0xD8
1	2	25	Shall not be used	0xD9
1	2	26	Shall not be used	0xDA
1	2	27	Shall not be used	0xDB
1	2	28	Error Alarm_Ack	0xDC
1	2	29	Shall not be used	0xDD
1	2	30	Error Read	0xDE
1	2	31	Error Write	0xDF

6.2.7.2 Coding of the Field Error_Decode

The coding of this field shall be as data type Unsigned8 and the values shall be set according to the table below:

Table 355 - Values of Error_Decode

0 to 127 =	reserved
128 =	DPV1
129 to 253 =	reserved
254 to 255 =	PROFILE_SPECIFIC

The field Error_Decode defines the meaning of the following Octets Error_Code_x. The Error_Decode values PROFILE_SPECIFIC indicate that the parameters Error_Code_x shall be set as defined in the corresponding protocols:

PROFILE_SPECIFIC

Error_Code_1= taken from profile specification

Error_Code_2= taken from profile specification

6.2.7.3 Coding of the Field Error_Code_1

If field Error_Decompose = DPV1

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 3: Error_Code

The values shall be set according to the Error_Code column in the table below.

Bit 4 to 7: Error_Class

The values shall be set according to the Error_Class column in the table below.

Table 356 - Coding of Error_Code_1 at DPV1

Error_Class (decimal)	Meaning	Error_Code (decimal)
0 to 9	not specified	not specified *)
10	application	0 = read error 1 = write error 2 = module failure 3 to 7 = not specified 8 = version conflict 9 = feature not supported 10 to 15 = User specific
11	access	0 = invalid index 1 = write length error 2 = invalid slot 3 = type conflict 4 = invalid area 5 = state conflict 6 = access denied 7 = invalid range 8 = invalid parameter 9 = invalid type 10 to 15 = User specific
12	resource	0 = read constrain conflict 1 = write constrain conflict 2 = resource busy 3 = resource unavailable 4 to 7 = not specified 8 to 15 = User specific
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.

If field Error_Decompose = PROFILE_SPECIFIC

Error_Code_1= from profile specification, needs to be specified in the profile definition

6.2.7.4 Coding of the Field Error_Code_2

If field Error_Decompose = DPV1

The coding of this field shall be as data type Unsigned8. The values are user specific.

If field Error_Decompose = PROFILE_SPECIFIC

Error_Code_2=from profile specification, needs to be specified in the profile definition

6.2.8 Coding Section related to Master Diagnosis PDU

6.2.8.1 Coding of the Field MDiag_Identifier

The coding of this field shall be as data type Unsigned8 and the values shall be set according to the table below:

Table 357 - Values of MDiag_Identifier

Value (decimal)	Meaning
0-125	Diag_Data of the DP-Slave
126	System_Diagnostis
127	Master_Status
128	Data_Transfer_List
129-255	reserved

6.2.8.2 Coding of the Field System_Diagnosis

The coding of this field shall be as data type Octet String with the fixed length of 16. The 16 octets of System_Diagnosis shall be coded as follows

Octet 1

Bit 0 shall be set to one if station number 0 has reported diagnosis otherwise it shall be set to zero.
 Bit 1 shall be set to one if station number 1 has reported diagnosis otherwise it shall be set to zero.
 Bit 2 shall be set to one if station number 2 has reported diagnosis otherwise it shall be set to zero.
 Bit 3 shall be set to one if station number 3 has reported diagnosis otherwise it shall be set to zero.
 Bit 4 shall be set to one if station number 4 has reported diagnosis otherwise it shall be set to zero.
 Bit 5 shall be set to one if station number 5 has reported diagnosis otherwise it shall be set to zero.
 Bit 6 shall be set to one if station number 6 has reported diagnosis otherwise it shall be set to zero.
 Bit 7 shall be set to one if station number 7 has reported diagnosis otherwise it shall be set to zero.

Octet 2

Bit 0 shall be set to one if station number 8 has reported diagnosis otherwise it shall be set to zero.
 Bit 1 shall be set to one if station number 9 has reported diagnosis otherwise it shall be set to zero.

etc.

Octet 16

Bit 5 shall be set to one if station number 125 has reported diagnosis otherwise it shall be set to zero.
 Bit 6 to 7 shall be set to zero (not used)

6.2.8.3 Coding of the Field USIF_State

This field shall be coded as data type Unsigned8. The following values are defined:

Value(0x40):shall be set if operation mode of DP-Master (Class 1) is Stop

Value(0x80):shall be set if operation mode of DP-Master (Class 1) is Clear

Value(0xC0): shall be set if operation mode of DP-Master (Class 1) is Operate

6.2.8.4 Coding of the Field Hardware_Release_DP

This field shall be coded as data type Unsigned8.

6.2.8.5 Coding of the Field Firmware Release_DP

This field shall be coded as data type Unsigned8.

6.2.8.6 Coding of the Field Hardware_Release_User

This field shall be coded as data type Unsigned8.

6.2.8.7 Coding of the Field Firmware Release_User

This field shall be coded as data type Unsigned8.

6.2.8.8 Coding of the Field Data_Transfer_List

The coding of this field shall be as data type Octet String with the fixed length of 16. The 16 octets of Data_Transfer_List shall be coded as follows

Octet 1

Bit 0 shall be set to one if data exchange executed with station number 0 otherwise it shall be set to zero.

Bit 1 shall be set to one if data exchange executed with station number 1 otherwise it shall be set to zero.

Bit 2 shall be set to one if data exchange executed with station number 2 otherwise it shall be set to zero.

Bit 3 shall be set to one if data exchange executed with station number 3 otherwise it shall be set to zero.

Bit 4 shall be set to one if data exchange executed with station number 4 otherwise it shall be set to zero.

Bit 5 shall be set to one if data exchange executed with station number 5 otherwise it shall be set to zero.

Bit 6 shall be set to one if data exchange executed with station number 6 otherwise it shall be set to zero.

Bit 7 shall be set to one if data exchange executed with station number 7 otherwise it shall be set to zero.

Octet 2

Bit 0 shall be set to one if data exchange executed with station number 8 otherwise it shall be set to zero.

Bit 1 shall be set to one if data exchange executed with station number 9 otherwise it shall be set to zero.

etc.

Octet 16

.
.

Bit 5 shall be set to one if data exchange executed with station number 125 otherwise it shall be set to zero.

Bit 6 to 7 shall be set to zero (not used)

6.2.9 Coding Section related to Upload/Download/Act Para PDUs**6.2.9.1 Coding of the Field Area_Code_UpDownload**

The coding of this field shall be as data type Unsigned8 and the values shall be set according to the table below.

Table 358 - Values

Value (decimal)	Meaning
0 to 125	DP-Slave parameter set (see field DP_Slave_Parameter_Set)
126	Shall not be used
127	Bus parameter set (see field Bus_Parameter_Set)
128	Shall not be used
129	statistic counters (see field Statistic_Counter)
130 to 135	Shall not be used
136 to 139	Master parameter set (see field Master_Parameter_Set)
140 to 254	Shall not be used
255	No local access protection in Start_Seq-REQ-PDU or Start_Seq-RES-PDU

6.2.9.2 Coding of the Field Timeout

The coding of this field shall be as data type Unsigned16 and the time base for the timer shall be 1ms.

6.2.9.3 Coding of the Field Max_Len_Data_Unit

The coding of this field shall be as data type Unsigned8 and the value for the Max_Len_Data_Unit shall be in the range from one to 240.

6.2.9.4 Coding of the Field Add_Offset

The coding of this field shall be as data type Unsigned16.

6.2.9.5 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, Octet String, Date, Time of Day, Time-Difference.

6.2.9.6 Coding of the Field Data_Len

The coding of this field shall be as data type Unsigned8 and the value for the Data_Len shall be in the range from one to 240.

6.2.9.7 Coding of the Field Area_CodeActBrct

The coding of this field shall be as data type Unsigned8 and the values shall be set according to the table below.

Table 359 - Value Range

Value (decimal)	Meaning
0 to 126	Shall not be used
127	Bus_parameter set
128 to 129	Shall not be used
130 to 135	Shall not be used
136 to 139	Master parameter set
140 to 254	Shall not be used
255	Shall not be used

6.2.9.8 Coding of the Field Area_CodeAct

The coding of this field shall be as data type Unsigned8 and the values shall be set according to the table below.

Table 360 - Value Range

Value (decimal)	Meaning
0 to 125	DP-Slave parameter set The Active Flag in the DP-Slave parameter set of the DP-Master (Class 1) is influenced accordingly. The DP-Slave concerned takes part in the cyclic data exchange mode or is removed from this mode and no longer addressed.
126	Shall not be used
127	Bus parameter set
128	operation mode
129	Shall not be used
130 to 135	Shall not be used
136 to 139	Master parameter set
140 to 255	Shall not be used

6.2.9.9 Coding of the Field Activate

This field shall be coded as data type Unsigned8. The following values are defined:

Value(0x80): means activate the corresponding DP-Slave parameter set if the field Area_CodeAct equals 0.. 125

Value(0x00): means deactivate if the field Area_CodeAct equals 0.. 125

Value(0xFF): means activate the bus parameter set if the field Area_CodeAct equals 127

Value(0x40): means set operation mode to Stop if the field Area_CodeAct equals 128

Value(0x80): means set operation mode to Clear if the field Area_CodeAct equals 128

Value(0xC0): means set operation mode to Operate if the field Area_CodeAct equals 128

6.2.10 Coding Section related to the Bus Parameter Set**6.2.10.1 Coding of the Field Bus_Para_Len**

This field shall contain the length of Bus_Para inclusive the field Bus_Para_Len itself. This field shall be coded as data type Unsigned16. The range of values shall be from 66 to $2^{16}-1$.

6.2.10.2 Coding of the Field DL_Add

This field shall be coded as data type Unsigned8. The range of values shall be zero to 125. This parameter shall contain the own address of the DP-Master.

6.2.10.3 Coding of the Field Baud_rate

This field shall be coded as data type Unsigned8. The values shall be set as follows:

Table 361 - Values for Baud_rate

Value (decimal)	Meaning
0	9,6kbit/s
1	19,2kbit/s
2	93,75kbit/s
3	187,5kbit/s
4	500 kbit/s
5	Shall not be used
6	1500 kbit/s
7	3000 kbit/s
8	6000 kbit/s
9	12000 kbit/s
10	31,25 kbit/s
11	45,45 kbit/s
12 to 255	Shall not be used

6.2.10.4 Coding of the Fields T_{SL} , min T_{SDR} , max T_{SDR}

These parameters are described in part 3. The coding of these fields shall be as data type Unsigned16.

6.2.10.5 Coding of the Fields T_{QUI} , T_{SET} , G , HSA , max_retry_limit

These parameters are described in part 3. The coding of these fields shall be as data type Unsigned8.

6.2.10.6 Coding of the Field T_{TR} (Target Token Rotation Time)

This field is described in part 3. The coding of this field shall be as data type Unsigned32.

6.2.10.7 Coding of the Field Bp_Flag (Busparameter Flag)

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 6: reserved

Bit 7: Error_Action_Flag

This bit shall be set to one if the DP Master (Class 1) shall change the operation mode in case of an error. Otherwise this bit shall be set to zero.

6.2.10.8 Coding of the Field Min_Slave_Interval

The coding of this field shall be as data type Unsigned16 with the value range from 1 to $2^{16}-1$ and the time base for the timer shall be 100µs.

6.2.10.9 Coding of the Field Poll_Timeout

The coding of this field shall be as data type Unsigned16 with the value range from 1 to $2^{16}-1$ and the time base for the timer shall be 1ms.

6.2.10.10 Coding of the Field Data_Control_Time

The coding of this field shall be as data type Unsigned16 with the value range from 1 to $2^{16}-1$ and the time base for the timer shall be 10ms.

6.2.10.11 Coding of the Field Alarm_Max

The coding of this field shall be as data type Unsigned8 with the value range from 7 to 32.

6.2.10.12 Coding of the Field Max_User_Global_Control

The coding of this field shall be as data type Unsigned8 with the value range from one to 255. Zero is reserved and shall not be used.

NOTE This parameter defines the maximum number of Global_Control requests, which may be started by the User at the same time. The parameter describes the ability of the DP-Master. A practicable value for Max_User_Global_Control is 16. For each of the 8 Slave groups one SYNC and one FREEZE command may be started at the same time.

6.2.10.13 Coding of the Field Master_User_Data_Len

This field shall contain the length of Master_User_Data inclusive the field Master_User_Data_Len itself. This field shall be coded as data type Unsigned16. The range of values shall be from 2 to ($2^{16}-1$).

6.2.10.14 Coding of the Field Master_Class2_Name

The coding of this field shall be as data type Visible String with the fixed length of 32.

6.2.10.15 Coding of the Field Master_User_Data

One of the following data types shall be used for each Master_User_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.11 Coding Section related to the Slave Parameter Set

6.2.11.1 Coding of the Field Slave_Para_Len

The coding of this field shall be as data type Unsigned16 and the value for the Slave_Para_Len shall be in the range from zero to $2^{16}-1$. This parameter shall contain the length of Slave_Para inclusive the length parameter itself. A DP-Slave parameter set can be deleted by setting the Slave_Para_Len to zero.

6.2.11.2 Coding of the Field SI_Flag (Slave Flag)

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: reserved

Bit 1: Extra_Alarm_SAP

This bit shall be set if the DP-Master (Class 1) shall acknowledge alarms via SAP 50. Otherwise, if the DP-Master (Class 1) acknowledges alarms via SAP 51 this bit shall not be set.

Bit 2: DPV1_Data_Types

This bit shall be set if extended data types are used in configuration (Extended Special Identifier Format). Otherwise, if only byte or word types with the Identifier_Format or Special Identifier Format this bit may not be set.

Bit 3: DPV1_Supported

This bit shall be set to enable the DPV1 extensions. Otherwise the extensions shall be disabled.

Bit 4: reserved

Bit 5: Fail_Safe

This bit shall be set to force the DP-Master (Class 1) to convey a NULL-PDU during operation mode Clear. Otherwise data with the value zero will be send during Clear.

Bit 6: New_Prm

This bit shall be set to force the DP-Master (Class 1) to convey a new Set_Prm-REQ-PDU during the data transfer phase. Otherwise this bit shall be set to zero.

Bit 7: Active

This bit shall be set to force the DP-Master (Class 1) to activate the corresponding DP-Slave. Otherwise this bit shall be set to zero.

6.2.11.3 Coding of the Field Slave_Type

This field shall be coded as data type Unsigned8 with values according to the table below:

Table 362 - Values for Slave_Type

Value (decimal)	Meaning
0	DP-Slave
1 to 15	Shall not be used
16 to 255	manufacturer specific

6.2.11.4 Coding of the Field Max_Diag_Data_Len

This field shall be coded as data type Unsigned8 and with values from the range 6 to 244.

6.2.11.5 Coding of the Field Max_Alarm_Len

This field shall be coded as data type Unsigned8 and with values from the range 4 to Min(Max_Diag_Data_Len-6, 64).

6.2.11.6 Coding of the Field Max_Channel_Data_Length

This field shall be coded as data type Unsigned8 and with values from the range 4 to 244.

NOTE This parameter defines the maximum APDU size for the corresponding DP-Slaves on the MS1-AR.

6.2.11.7 Coding of the Field Diag_Upd_Delay

This field shall be coded as data type Unsigned8 and with values from the range 0 to 15 (extendible up to 255).

NOTE The parameter is used to count the number of Slave_Diag.cnf in the state DIAG2 while Diag_Data.Prm_Req is still set (for Slaves with reduced performance).

6.2.11.8 Coding of the Field Alarm_Mode

This parameter specifies the maximum number of possible active alarms.

This field shall be coded as data type Unsigned8 with values according to the table below:

Table 363 - Values for Alarm_Mode

Value (decimal)	Meaning
0	1 alarm of each type
1	2 alarms in total
2	4 alarms in total
3	8 alarms in total
4	12 alarms in total
5	16 alarms in total
6	24 alarms in total
7	32 alarms in total

6.2.11.9 Coding of the Field Add_SI_Flag

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: NA_To_Abort

This bit shall be set if the DP-Master (Class 1) shall proceed with data exchange in case the corresponding DP-Slave is not responding. Otherwise this bit shall not be set.

Bit 1: Ignore_ACIr

This bit shall be set if the DP-Master (Class 1) shall exclude the corresponding DP-Slave from the auto-clear behaviour. Otherwise this bit shall not be set.

Bit 2 to 7: reserved**6.2.11.10 Coding of the Field Prm_Data_Len**

This parameter contains the length of Prm_Data inclusive the length parameter.

This field shall be coded as data type Unsigned16 and with values from the range 9 to 246.

6.2.11.11 Coding of the Field Prm_Data

This field shall be coded as a Set_Prm-REQ-PDU.

6.2.11.12 Coding of the Field Cfg_Data_Len

This parameter contains the length of Cfg_Data inclusive the length parameter.

This field shall be coded as data type Unsigned16 and with values from the range 3 to 246.

6.2.11.13 Coding of the Field Cfg_Data

This field shall be coded as a Chk_Cfg-REQ-PDU.

6.2.11.14 Coding of the Field Add_Tab_Len

This parameter contains the length of Add_Tab inclusive the length parameter.

This field shall be coded as data type Unsigned16 and with values from the range 2 to $2^{16}-31$.

6.2.11.15 Coding of the Field Number_of_Entries

This field contains the number of entries in the address table.

This field shall be coded as data type Unsigned16 and with values from the range 1 to 488.

NOTE This parameter contains the number of entries in the address assignment table in the host to the addressed DP-Slave. At maximum 488 entries per DP-Slave are allowed (maximal 244 configuration strings doubled for inputs and outputs are possible).

6.2.11.16 Coding of the Field Add_Tab_Entry_Header

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: IO_Selection

This bit shall be set if the following addresses belong to the output part. Otherwise this bit shall not be set and the following addresses belong to the input part.

Bit 1: Address_Format_Selection

This bit shall be set to one if the following addresses are coded as Unsigned32. This bit shall be set to zero if the following addresses are coded as Unsigned16.

Bit 2 to 7: reserved**6.2.11.17 Coding of the Field I/O_Data_Length**

This field shall be coded as data type Unsigned8 with the values from one to 244.

6.2.11.18 Coding of the Field I/O_Config_Address

This field represents the local address of the related configuration identifier.

This field shall be coded as data type Unsigned16 if the bit Address_Format_Selection in the field Add_Tab_Entry_Header is set to zero.

This field shall be coded as data type Unsigned32 if the bit Address_Format_Selection in the field Add_Tab_Entry_Header is set to one.

6.2.11.19 Coding of the Field Host_Address

This field represents the local address of the related input or output data.

This field shall be coded as data type Unsigned16 if the bit Address_Format_Selection in the field Add_Tab_Entry_Header is set to zero.

This field shall be coded as data type Unsigned32 if the bit Address_Format_Selection in the field Add_Tab_Entry_Header is set to one.

6.2.11.20 Coding of the Field Slave_User_Data_Len

This parameter contains the length of Slave_User_Data inclusive the length parameter.

This field shall be coded as data type Unsigned16 and with values from the range 2 to $2^{16}-31$.

6.2.11.21 Coding of the Field Slave_User_Data

One of the following data types shall be used for each Slave_User_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.12 Coding Section related to Statistic Counters

6.2.12.1 Coding of the Field Frame_sent_count and SD_count

The coding of this field shall be as data type Unsigned32.

6.2.12.2 Coding of the Field Error_count and SD_error_count

The coding of this field shall be as data type Unsigned16.

6.2.13 Coding Section related to Set Slave Address PDU

6.2.13.1 Coding of the Field New_Slave_Add

This field shall be coded as data type Unsigned8 with values from the range zero to 125.

6.2.13.2 Coding of the Field No_Add_Change

This field shall be coded as data type Boolean. The value TRUE shall be used to indicate that the change of the address is only once possible. The value FALSE shall be used to indicate that the change of the address is more than once possible.

6.2.13.3 Coding of the Field Rem_Slave_Data

One of the following data types shall be used for each Rem_Slave_Data field:

Boolean, Integer, Unsigned, Floating Point, Visible String, Octet String, Date, Time of Day, Time-Difference.

6.2.14 Coding Section related to Initiate/Abort PDUs

6.2.14.1 Coding of the Field Features_Supported_1

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0: Read_Write

This bit shall be set to indicate that Read and Write services on MS2-AR are supported.

NOTE Read and Write are mandatory on a MS2-AR.

Bit 1 to 7: reserved

6.2.14.2 Coding of the Field Features_Supported_2

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 7: reserved

6.2.14.3 Coding of the Field Profile_Features_Supported_1

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 7: defined by profile

This field shall be set to zero (decimal) if no profile is used. Otherwise the value shall be taken from the appropriate profile specification.

6.2.14.4 Coding of the Field Profile_Features_Supported_2

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 7: defined by profile

This field shall be set to zero (decimal) if no profile is used. Otherwise the value shall be taken from the appropriate profile specification.

6.2.14.5 Coding of the Field Profile_Ident_Number

The coding of this field shall be as data type Unsigned16. The value zero shall be set if no profile is used. Otherwise the value shall be taken from the appropriate profile specification.

6.2.14.6 Coding of the Field S_Type (Source Type)

The coding of this field shall be as data type Unsigned8. The value zero shall be set to indicate the use of the short address format. The value one shall be set to indicate the use of the long address format. Other values shall not be used.

NOTE The terms source and destination always belong to the direction of the used PDU.

6.2.14.7 Coding of the Field D_Type (Destination Type)

The coding of this field shall be as data type Unsigned8. The value zero shall be set to indicate the use of the short address format. The value one shall be set to indicate the use of the long address format. Other values shall not be used.

6.2.14.8 Coding of the Field S_Len (Source Length)

This field shall be coded as data type Unsigned8.

6.2.14.9 Coding of the Field D_Len (Destination Length)

This field shall be coded as data type Unsigned8.

6.2.14.10 Coding of the Field S_API (Source Application Identifier)

This field shall be coded as data type Unsigned8.

6.2.14.11 Coding of the Field D_API (Destination Application Identifier)

This field shall be coded as data type Unsigned8.

6.2.14.12 Coding of the Field S_SCL (Source Security Level)

This field shall be coded as data type Unsigned8. The value zero shall be set to indicate that no access level is used.

6.2.14.13 Coding of the Field D_SCL (Destination Security Level)

This field shall be coded as data type Unsigned8. The value zero shall be set to indicate that no access level is used.

6.2.14.14 Coding of the Field S_Network_Address

The field shall only be present if the field S_Type is set to one.

The coding of this field shall be as data type Octet String with the fixed length of 6.

6.2.14.15 Coding of the Field D_Network_Address

The field shall only be present if the field D_Type is set to one.

The coding of this field shall be as data type Octet String with the fixed length of 6.

6.2.14.16 Coding of the Field S_MAC_Address

The field shall only be present if the field S_Type is set to one.

The coding of this field shall be as data type Octet String.

6.2.14.17 Coding of the Field D_MAC_Address

The field shall only be present if the field D_Type is set to one.

The coding of this field shall be as data type Octet String.

6.2.14.18 Coding of the Field Send_Timeout

This field shall be coded as data type Unsigned16. The time base for the timer shall be 10 ms.

6.2.14.19 Coding of the Field Server_SAP

This field shall be coded as data type Unsigned8. The values shall be set from the range zero to 48.

6.2.14.20 Coding of the Field Subnet

The coding of this field shall be as data type Unsigned8 and the values shall be set according to the table below.

Table 364 - Values

Value (decimal)	Meaning
0	means no subnet
1	means subnet local
2	means subnet remote
3 .. 255	reserved

6.2.14.21 Coding of the Field Instance_Reason_Code

The coding of this field shall be according to 3.5.3.3 and the individual bits shall have the following meaning:

Bit 0 to 3: Reason Code

The values shall be set according to the table below if Instance is set to DLL.

Table 365 - Values of Reason Code if Instance is DLL

Value (decimal)	Meaning
1	UE (meaning see Part 3)
2	RR (meaning see Part 3)
3	RS (meaning see Part 3)
9	NR (meaning see Part 3)
10	DH (meaning see Part 3)
11	LR (meaning see Part 3)
12	RDL (meaning see Part 3)
13	RDH (meaning see Part 3)
14	DS (meaning see Part 3)
15	NA (meaning see Part 3)

The values shall be set according to the table below if Instance is set to MS2.

Table 366 - Values of Reason Code if Instance is MS2

Value (decimal)	Meaning
1	ABT_SE means sequence error
2	ABT_FE means invalid request APDU received
3	ABT_TO means connection timed out
4	ABT_RE means invalid response APDU received
5	ABT_IV means invalid service from the User
6	ABT_STO means requested value of Send_Timeout was too short
7	ABT_IA means invalid additional address information
8	ABT_OC means S-Timer expired, response APDU has not been sent yet

Bit 4 to 5: Instance

The values shall be set as follows:

Decimal (0): DLL

Decimal (1): MS2

Decimal (2): USER

Decimal (3): reserved, shall not be used

Bit 6 to 7: reserved**6.2.15 Coding Section related to Read/Write/Data Transport PDUs****6.2.15.1 Coding of the Field Index**

This field shall be coded as data type Unsigned8 with values from the range zero to 254. The value 255 is reserved and shall not be set.

6.2.15.2 Coding of the Field Length

This field contains the number of octets of the Data field.

This field shall be coded as data type Unsigned8 with values from the range zero to 240.

6.2.16 Example of Diagnosis-RES-PDU

msb							lsb
7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	1
1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1
0	1	0	1	0	0	0	0
0	1	0	0	0	0	1	0
0	0	0	0	0	1	0	1

Headerbyte: device related diagnostic
Status_Type: Status_Message
Slot_Number: 2 (sensor A)
Specifier: no further differentiation
Diagnostic_User_Data: 5 (average temperature)
Diagnostic_User_Data: temperature value (Unsigned16)
Headerbyte: device related diagnostic
Alarm_Type: Process_Alarm
Slot_Number: 3 (valve B)
Specifier: alarm appears
Diagnostic_User_Data: 0x50 (upper limit exceeded pressure)
Diagnostic_User_Data: time stamp (Time_of_day = 4 octets)
Headerbyte: identifier related diagnostic
1st identifier Number with diagnosis

Figure 115 - Example of Ext_Diag_Data in case of DPV1 diagnosis format with Alarm and Status PDU

Corresponding textual part

;text assignments for sensor A and valve B

Unit_Diag_Area = 24-27

Value(1) = "Minimum temperature"

Value(2) = "Maximum temperature"

Value(5) = "Average temperature"

Unit_Diag_Area_End

Unit_Diag_Area = 28-31

Value(1) = "lower limit exceeded pressure"

Value(5) = "upper limit exceeded pressure"

Unit_Diag_Area_End

Unit_Diag_Area = 8-15

Value(2) = "sensor A"

Value(3) = "valve B"

Unit_Diag_Area_End

Unit_Diag_Area = 16-17

Value(1) = "alarm/status appearing"

Value(2) = "alarm/status disappearing"

Unit_Diag_Area_End

Since these definitions are used for both alarms and status messages their values should be different. That means different values for alarms and status messages should be used at the same position within the diagnostic field.

msb				lsb			
7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0
device specific							
diagnostic field							
of length 3							
0	1	0	0	0	1	0	1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	1	0	0	1	0	0
1	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
1	0	1	0	0	1	1	1

device related diagnostic:
meaning of the bits is defined
manufacturer specific

identifier related diagnostic:
identifier number 0 has diagnostic
identifier number 12 has diagnostic
identifier number 18 has diagnostic

channel related diagnostic:
identifier number 0
channel 2
overload, channel bit organised
identifier number 12
channel 6
upper limit value exceeded, channel word organised

Figure 116 - Example of Ext_Diag_Data in case of the basic diagnosis format

6.2.17 Example of Chk_Cfg-REQ-PDU

Octet 1	1	1	0	0	0	0	1	1	input/output, 3 bytes manufacturer specific data
Octet 2	1	1	0	0	1	1	1	1	consistency, output, 16 words
Octet 3	1	1	0	0	0	1	1	1	consistency, input, 8 words
Octet 4	manufacturer								
Octet 5	specific								
Octet 6	data								

Figure 117 - Example of a special identifier format

6.2.18 Example of Chk_Cfg-REQ-PDU with Extension

The data types are coded into the manufacturer-specific part.

Simple data types are coded in a byte which contains the code of the data type.

Sequences of data types are coded as a list of simple data types. The sum of the lengths of the data types has to correspond to the input or output data length. Data types with variable length (visible string, octet string, time of day, time difference) cannot be handled in a sequence, but as single element.

The example below shows the special identification format for an analogue input block containing a value (Floating Point) and Status (Unsigned8).

Octet 1	0	1	0	0	0	0	1	0	Input, 2 Bytes
Octet 2	1	0	0	0	0	1	0	0	Manufacturer-specific format
Octet 3	0	0	0	0	1	0	0	0	Consistency, 5 Bytes (Input)
Octet 4	0	0	0	0	0	1	0	1	Data type Floating Point
									Data type Unsigned8

Figure 118 - Example special identification format

Example 2 shows the special identification format for an analogue output block where the output value can be read back.

The representation of this analogue output block is structured as follows:

Output Value (Floating Point)

Output Status (Unsigned8)

read back Value (Floating Point)

read back Status (Unsigned8)

Octet 1	1	1	0	0	0	1	0	0	Output/Input, 4 Bytes
Octet 2	1	0	0	0	0	1	0	0	Manuf.-specific format
Octet 3	1	0	0	0	0	1	0	0	Consistency, 5 Bytes (Output)
Octet 4	0	0	0	0	1	0	0	0	Consistency, 5 Bytes (Input)
Octet 5	0	0	0	0	0	1	0	1	Data Type Floating Point
Octet 6	0	0	0	0	1	0	0	0	Data Type Unsigned8
Octet 7	0	0	0	0	0	1	0	1	Data Type Floating Point
									Data Type Unsigned8

Figure 119 - Example special identification format

6.2.19 Example Structure of the Data_Unit for Data_Exchange

The structure of the Data_Unit for the service Data_Exchange is defined by the identifiers which are passed to the DP-Slave with the service Chk_Cfg at configuration.

The following example shows how the data in the Data_Unit will be transferred, corresponding to the given identifiers.

1. ID	2. ID	3. ID	4. ID	5. ID	6. ID	7. ID	...
1 B_I	1 W_I	2 W_O	2 B_IO	1 B_O	2 W_I	3 W_A	...

Figure 120 - Identifiers (ID)

1 B_I:	1 Byte input;
1 B_O:	1 Byte output;
2 B_IO:	2 Byte out- and input
1 W_I:	1 Word input
1 W_O:	1 Word output
etc.	

Figure 121 - Identifier list

The sequence of data in the Data_Unit is shown Figure 122.

Data_Exchange.req									Data_Exchange.res												
lsb									lsb												
Bit	8	7	6	5	4	3	2	1	Bit	8	7	6	5	4	3	2	1				
Byte 1	Word_high				(3. ID)				Byte 1	Data								(1. ID)			
2	Word_low				(3. ID)				2	Word_high				(2. ID)							
3	Data				(4. ID)				3	Word_low				(2. ID)							
4	Data				(4. ID)				4	Data				(4. ID)							
5	Data				(5. ID)				5	Data				(4. ID)							
6	1.Word_high				(7. ID)				6	1.Word_high				(6. ID)							
7	1.Word_low				(7. ID)				7	1.Word_low				(6. ID)							
8	2.Word_high				(7. ID)				8	2.Word_high				(6. ID)							
9	2.Word_low				(7. ID)				9	2.Word_low				(6. ID)							
10	3.Word_high				(7. ID)				10	etc.											
11	3.Word_low				(7. ID)				11												
12	etc.								12												
...									...												
n									n												

Figure 122 - Structure of the Data_Unit for the Request- and Response-Frame

For the configuration identifier free place no data will be transferred in the Data_Unit.

6.3 FAL Protocol State Machines

6.3.1 Overall Structure

6.3.1.1 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.

There exist one state machine for each device type (FSPMS for DP-Slave, FSPMM1 for DP-Master (Class 1) and FSPMM2 for DP-Master (Class 2).

6.3.1.2 Master to Slave Cyclic (MS0)

The MSCY1 State Machines are responsible for the cyclic data transfer between the DP-Master (Class 1) and one DP-Slave (MSCY1M/MSCY1S). This compromise the parameterisation, configuration, diagnostic and the data exchange as well as Global Control commands. There exist one MSCY1S for a DP-Slave and up to 125 MSCY1M for a DP-Master (Class 1).

6.3.1.3 Master (Class 1) to Slave Acyclic (MS1)

The MSAC1 State Machines (MSAC1M, MSAC1S) are responsible for the acyclic data transfer between the DP-Master (Class 1) and one DP-Slave. The MS1 communication relationship is coupled to the cyclic communication relationship and processes the services Read, Write and Alarm Ack.

Additionally the MSAL1M state machine is used for alarm handling at DP-Masters site.

There exist one MSAC1S for a DP-Slave and up to 125 MSAC1M/MSAL1M for a DP-Master (Class 1).

6.3.1.4 Master (Class 2) to Slave Acyclic (MS2)

The MS2 State Machines are responsible for the acyclic data transfer between the DP-Master (Class 2) and one DP-Slave. The MS2 communication relationship processes the services Read, Write and Data_Transport.

The Master Slave Resource Manager (MSRM2S) is responsible for the assignment of a SAP to one to be established MSAC2 communication relationship in the DP-Slave. The Master Slave Resource Manager links an incoming Initiate-Request to a SAP resource and starts the appropriate MSAC2S State Machine.

At DP-Masters site an arbitrary number of MSAC2M state machines exist (at most 125×49 Machines (= Number of Slaves * Number of DP-Slaves), can be active at any moment of time). At DP-Slave site, a maximum of 49 state machines may exist.

6.3.1.5 Master Master Acyclic (MM1/MM2)

By means of the MMAC State Machines the Master_Diag, the Master Parameter Set, Slave Parameter Set, Bus Parameter Set can be transferred between the DP-Master (Class 1) and the DP-Master (Class 2).

There exist at most one state machine at DP-Master (Class 1) and one at DP-Master (Class 2).

6.3.1.6 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 and DLM 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.

There exist one state machine for each device type (FSPMS for DP-Slave, FSPMM1 for DP-Master (Class 1) and FSPMM2 for DP-Master (Class 2).

Interface between DMPM and Layer 2

The DMPM is directly put upon the DLL User - DLL interface and on the DLMS User - DLM interface as described in Type 3 Data Link Layer Service Definitions (Refer to FAL Type 3 Service Definition IEC 61158-3).

The DMPM uses for the mapping of the DMPM functions on the Layer 2 the following services.

a) DLL services

Send and Request Data with Reply (SRD)

Send Data with No Acknowledge (SDN)

b) DLM services

Reset

Set Value

Read Value

Event

SAP Activate

SAP Activate Responder

SAP Deactivate

6.3.2 Assignment of State Machines to Devices

The following table shows the assignment of State Machines to devices. Some State Machines are always required and thus marked with 'M' (mandatory) others are marked with 'O'(optional). The Remarks show options in mandatory machines.

Table 367 - Assignment of State Machines

	Machine		Remarks
DP-Slave	FSPMS	M	Alarm handling is optional DPV1 = False mandatory. DPV1 = False may be omitted
	MSCY1S	M	
	MSAC1S	O	
	MSAC2S	O	
	MSRM2S	O	
	DMPMS	M	
DP-Master (Class 1)	FSPMM1	M	Master Diag service mandatory other services may be omitted
	MSCY1M	M	
	MSAL1M	O	
	MSAC1M	O	
	MMAC1	M	
	DMPMM1	M	
DP-Master (Class 2)	FSPMM2	M	
	MSAC2M	O	
	MMAC2	O	
	DMPMM2	M	

6.3.3 Overview DP-Slave

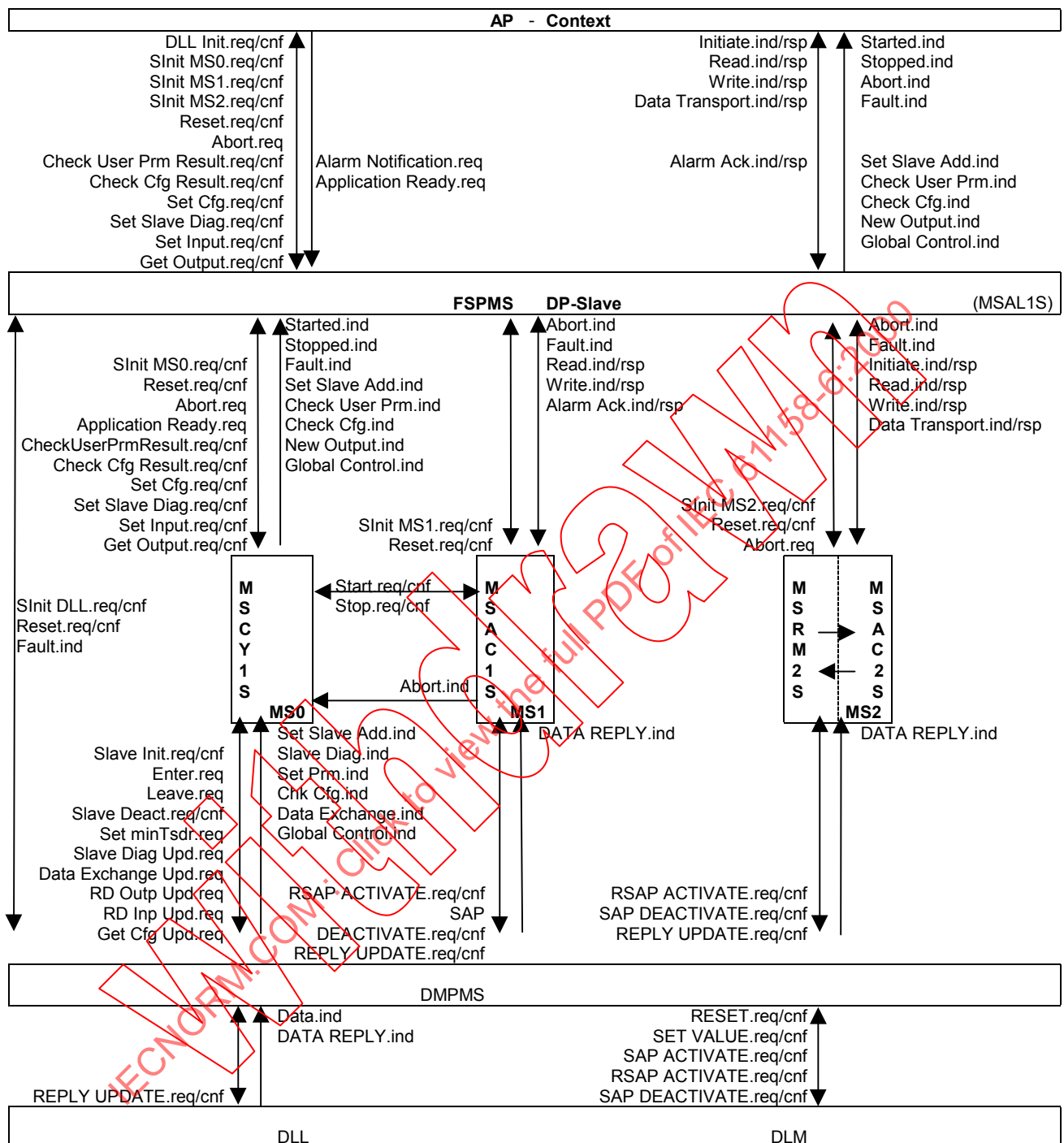


Figure 123 - Structuring of the Protocol Machines and Adjacent Layers in a DP-Slave

6.3.4 Overview DP-Master (Class 1)

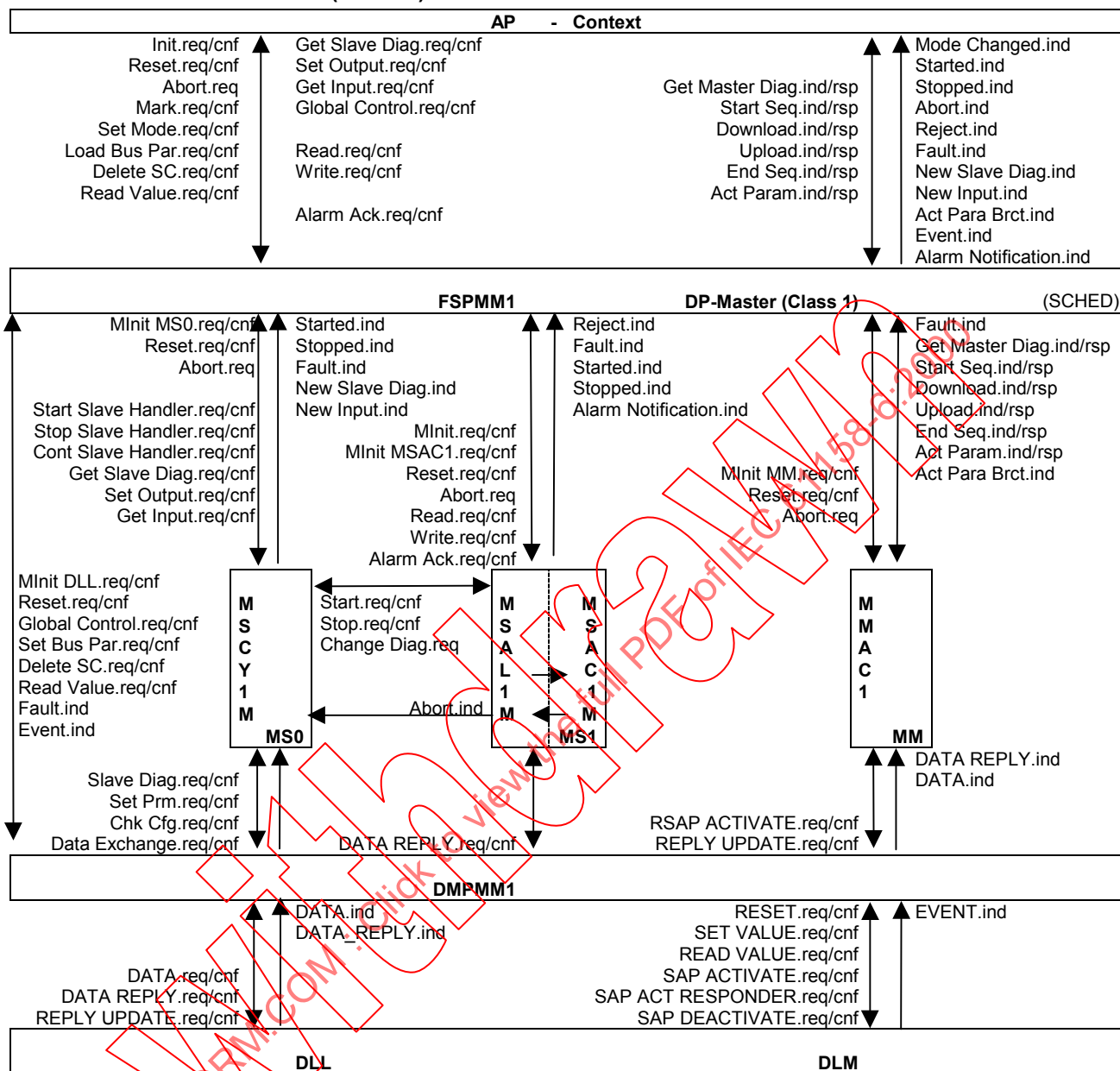


Figure 124 - Structuring of the Protocol Machines and Adjacent Layers in a DP-Master (Class 1)

6.3.5 Overview DP-Master (Class 2)

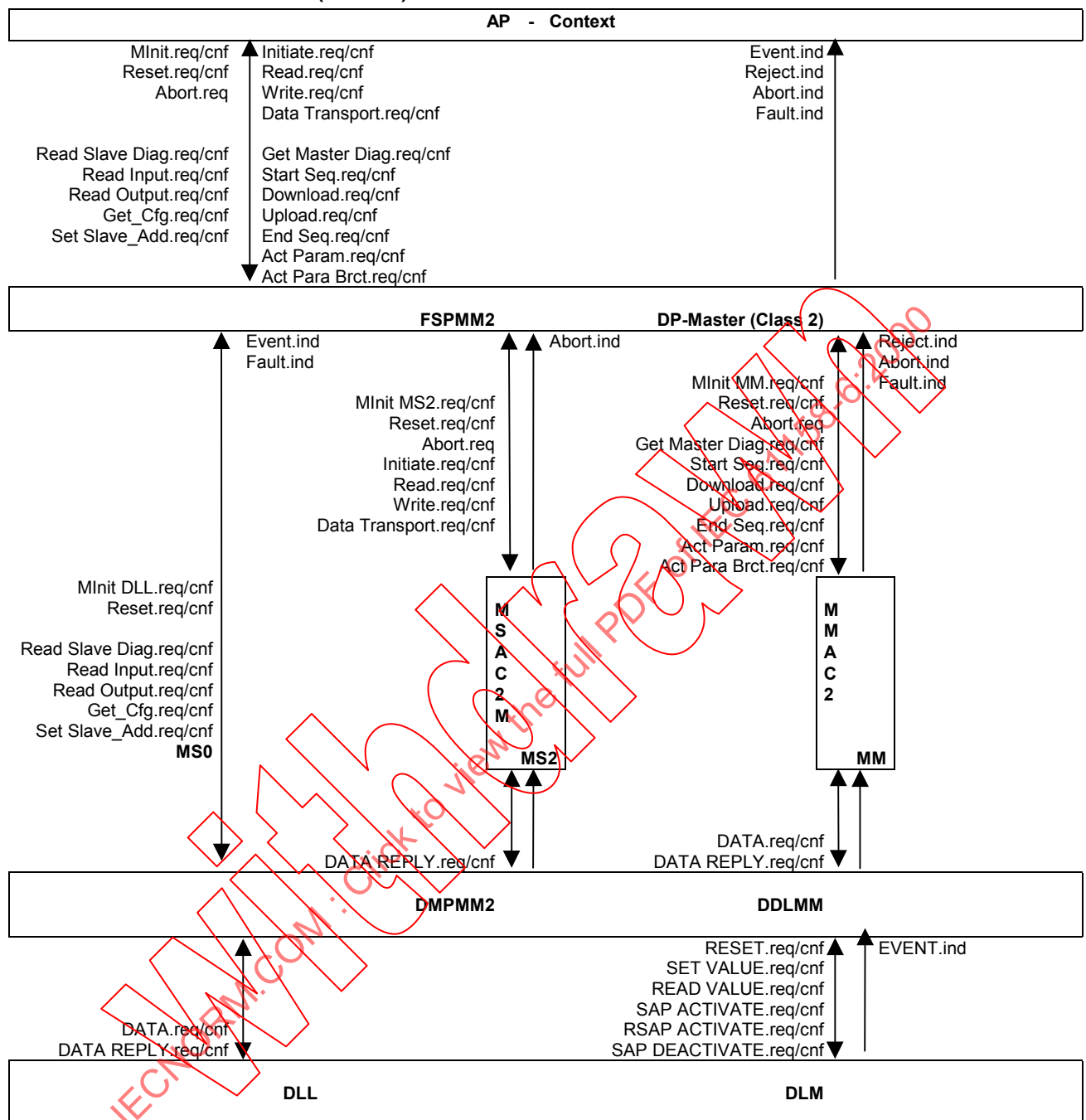


Figure 125 - Structuring of the Protocol Machines and Adjacent Layers in a DP-Master (Class 2)

6.3.6 Cyclic Communication between DP-Master (Class 1) and DP-Slave

The communication between DP-Master and DP-Slave is described in Figure 126.

If the DP-Master wants to communicate with a DP-Slave, the DP-Master requests the Diag_Data of the DP-Slave, in order to check the readiness for operation of the DP-Slave.

This Diagnosis request will be repeated until the DP-Slave responds with the requested data.

If the DP-Slave responds with the requested diagnostic information, the DP-Master checks whether another DP-Master occupies this DP-Slave. If this is not the case, the DP-Slave will be initialised and configured (Set_Prm / Chk_Cfg).

Subsequently the DP-Master requests again the diagnostic data from the DP-Slave. The following messages can occur:

- a) parameterisation or configuration error
- b) The DP-Slave is occupied by another DP-Master
- c) Static User diagnostics, no normal data transfer possible
(for Example: start-up procedure for DP correctly executed, but some external parameterisation outstanding)
- d) DP-Slave is not ready for operation

In the cases a) and b) the DP-Master returns to the starting state and checks again the readiness for operation of the DP-Slave.

In the cases c) and d) the DP-Master requests the diagnostic information from the DP-Slave until the mentioned messages disappear.

If no error messages occur the DP-Master starts the data exchange to the DP-Slave. The input and output data will be transferred to and from the DP-Slave. The DP-Master (Class 1) indicates his own operation mode to the DP-Slaves for the first time.

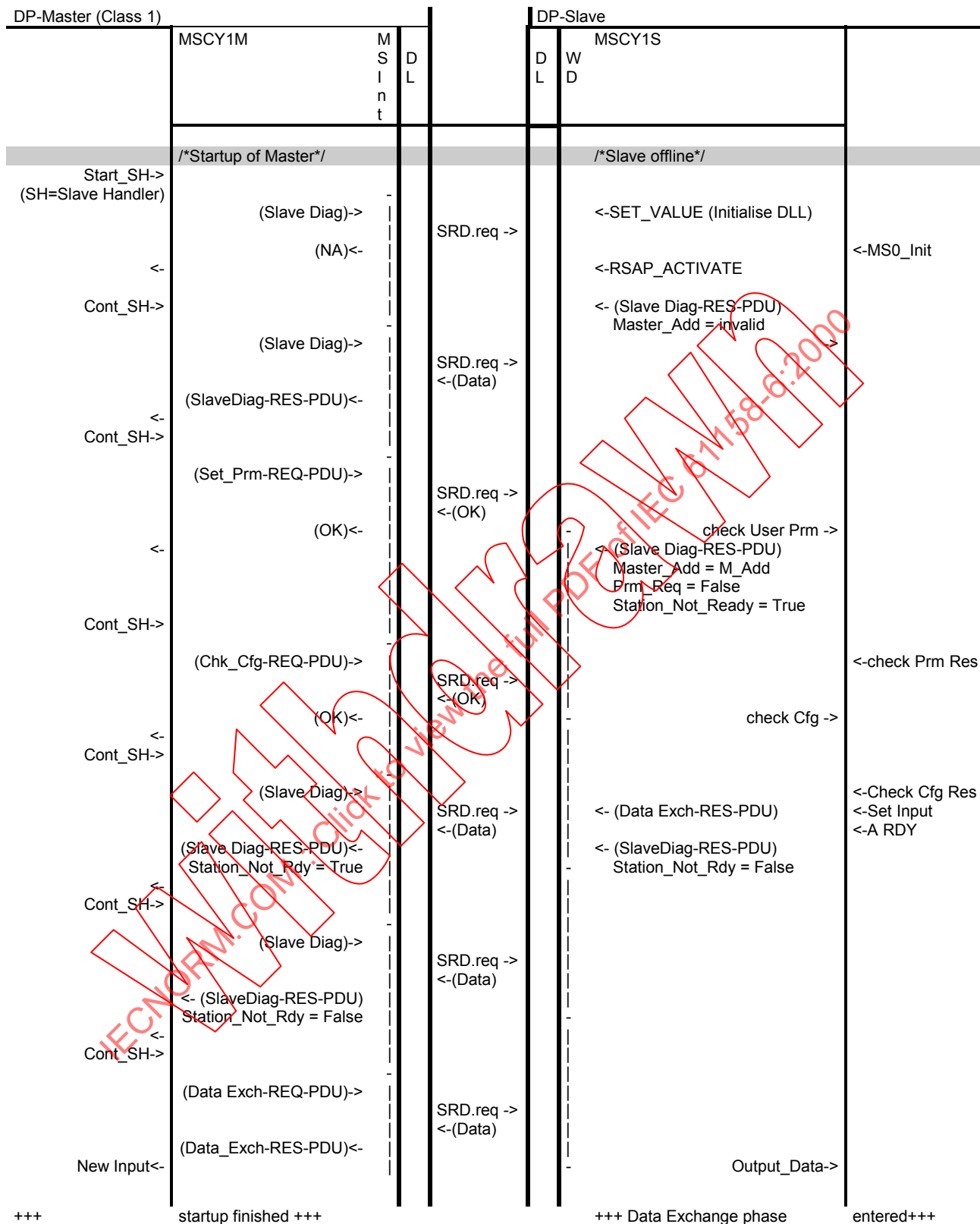


Figure 126 - Sequence of the communication between DP-Master and DP-Slave

6.3.7 Acyclic Communication between DP-Master (Class 2) and DP-Master (Class 1)

The communication between DP-Master (Class 2) and DP-Master (Class 1) is handled in a fixed way (see Figure 127) with one exception.

This exception is the function Act_Para_Brct, which is transferred as a multicast from the DP-Master (Class 2) to the corresponding DP-Master (Class 1).

Communication is always initiated by the DP-Master (Class 2). The DP-Master (Class 2) sends a request to the DP-Master (Class 1) and waits for the response. The service is processed by the MMAC2 of the DP-Master (Class 2). After the receipt of the response the DP-Master (Class 2) can make a new request.

At one point of time the DP-Master (Class 1) can only communicate with one DP-Master (Class 2).

The DP-Master communication is initiated by a xxx.req from the DP-Master (Class 2). The MMAC1 passes this request to Layer 2 and checks the status after receipt of the DMPM-response. If the response status is NR (no reply data) it will continue with a new SRD with no L_sdu to get response data. This procedure will be repeated by the DP-Master (Class 2) until

- an erroneous response occurs
- or a SRD_RES_PDU with response data is received
- or the Timer T1 expires.

The result will then be passed to the User of the DP-Master (Class 2) by means of a corresponding xxx.cnf.

The timer T1 will be started after receipt of the xxx.req in the MMAC2 of the DP-Master (Class 2) and will be stopped after release of the xxx.cnf. The value loaded in the timer T1 depends on the DLL parameter and the performance of the DP-Master (Class 1).

The responder (DP-Master (Class 1)) waits after a reset for a request. If the MMAC1 of the DP-Master (Class 1) receives a valid request, this request is passed to the User by the corresponding xxx.ind. At this point of time an access protection was set to the local service access point (SAP) for the communication with the DP-Master (Class 2). This is necessary to make sure that the response data will be sent to the right DP-Master (Class 2).

After receipt of the response from the User (xxx.rsp) the response data will be passed with a REPLY_UPDATE.req to DMPM. Additionally, a timer will be started which controls the request for the response. If this timer expires and the response data are still not fetched by the DP-Master (Class 2), the MMAC1 deletes the response data because the MMAC1 presumes that the DP-Master (Class 2) is no longer available. In this case the access protection for the DP-Master (Class 2) will be cancelled. This will also be done in the case of a single transaction if a DATA_REPLY.ind indicates that the response data are fetched by the DP-Master (Class 2). The value loaded in the timer of the DP-Master (Class 1) depends on the DL parameter and the performance of the DP-Master (Class 2).

In case of Master-Master communication a sequence of requests can put into parentheses. Therefore the access protection will be set by the "Start_Seq" and released by the "End_Seq". To avoid errors the time between two requests will be supervised. The value for the time supervision is added to the Start_Seq request. With the Start_Seq.req the User can additionally send an areacode. By means of this areacode at the beginning of a sequence an area can be reserved at the User in the DP-Master (Class 1).

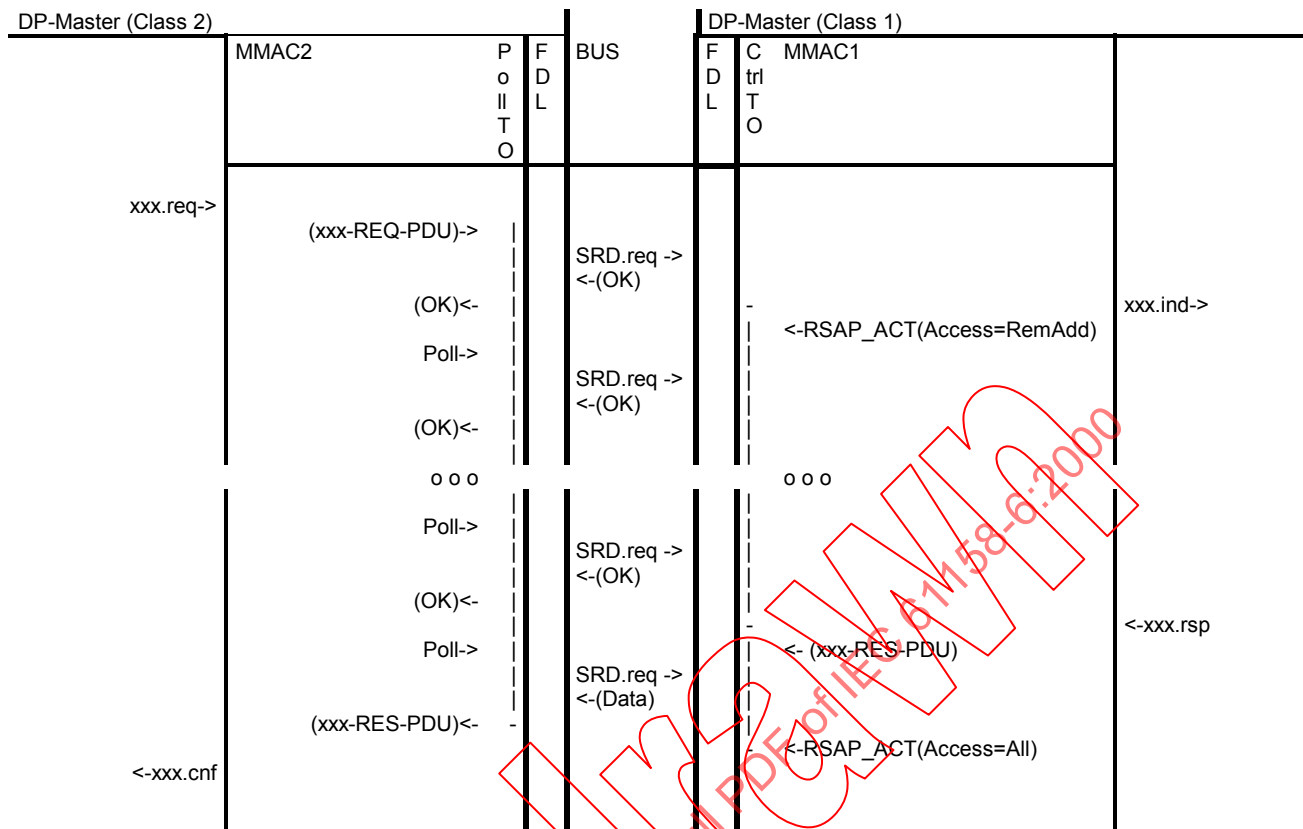


Figure 127 - Sequence of communication between DP-Master (Class 2) and DP-Master (Class 1)

6.3.8 Application Relationship Monitoring

6.3.8.1 Monitoring of the MS0 - AR

In industrial control systems there is a need for checking the correct functioning of each individual part. In a such a system a medium failure or a defect in a bus master shall not cause errors in the peripheral devices. In the above mentioned cases, a peripheral devices shall switch to a safe state. Additionally, the User of a bus master (Class 1) has to be informed about transmission breakdowns after a certain amount of time.

Control Interval at the DP-Slave

Watchdog control

This Timer, restarted by received Requests on the bus master side, will set the outputs of a bus slave to the safe state after the expiration of the timer.

Control Intervals at the DP-Master (Class 1)

Min_Slave_Interval

This monitoring time specifies the smallest allowed period of time between two Slave poll cycles. This ensures that the sequence of function requests from the bus master can be handled by the bus slave. This period of time will be complied by the bus master (Class 1) for every master-slave function with exception of the function Global_Control. For the function Global_Control the User is responsible for compliance with Min_Slave_Interval.

Data_Control_Time

The data transfer between the DP-Master (Class 1) and its associated DP-Slaves will be controlled by the corresponding state machines. By means of this a DP-Master (Class 1) checks the data transfer to its DP-Slaves.

Additionally the Data_Control_Time defines the time in which a DP-Master (Class 1) indicates his operation mode to the dedicated DP-Slaves.

The Master parameter set contains the Data_Control_Time.

Rule:

$$T_{WD} > T_{TR} \quad (1)$$

$$T_{WD} > \text{Min\_Slave\_Interval} \quad (2)$$

$$\text{Data\_Control\_Time} \geq 6 \cdot T_{WD} \quad (3)$$

6.3.8.2 Monitoring of the MS2 - AR

The objective of the connection monitoring is to detect the failure of the communication partner. Idle PDUs are exchanged to retrigger the monitoring timer of the communication partner.

The Idle service cycle initiated by a bus master is called Master-Idle. The Idle service cycle initiated by the Slave is called Slave-Idle.

All monitoring and idle sequences are controlled with a single timebase "Send Timeout".

Send_Timeout

Each MSAC2 communication relationship is supervised by a Timer (Send_Timeout).

If this Timer expires the bus master (Class 2) aborts the MSAC2-communication relationship. Additionally the User of a bus master (Class 2) has to be informed.

This Timer has to be set up according to

- the network load
- the number of parallel connections per DP-Master

Rule

$$T_{\text{Send\_TO}}(TS) \geq (\sum \text{of all Activities of a Token cycle}) \cdot \text{Num\_of\_max acyclic connections}(TS) \quad (4)$$

NOTE

The Send_Timeout parameter in the Initiate service is coded as an unsigned integer value, each unit represents 10ms. Due to Timer errors and the round up of integer, the next possible integer plus one has to be used as Send_Timeout.

It is assumed that one acyclic interaction is executed within a Token cycle at each DP-Master. If more than one can be executed, the Send_TO may be divided by the number of acyclic interactions per token cycle.

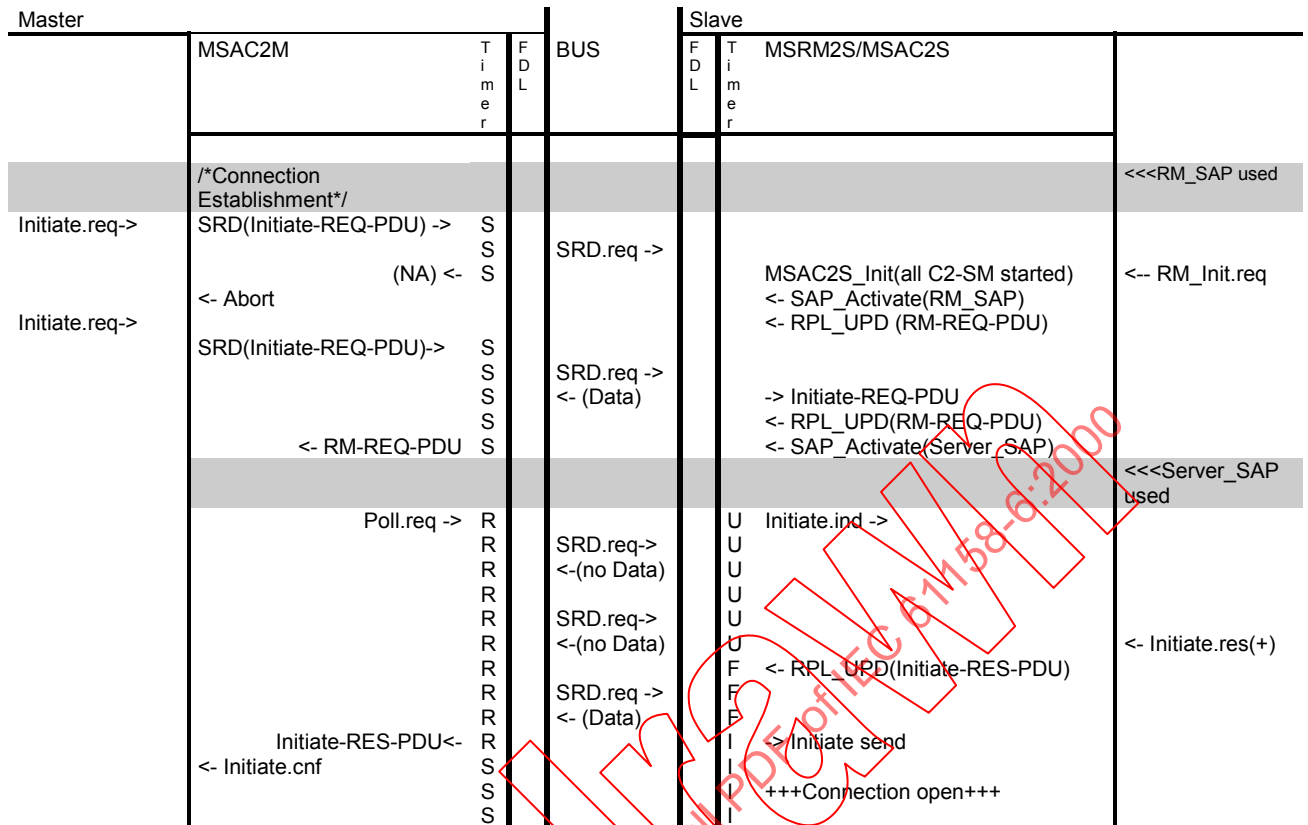


Figure 128 - Example for connection establishment on MS2

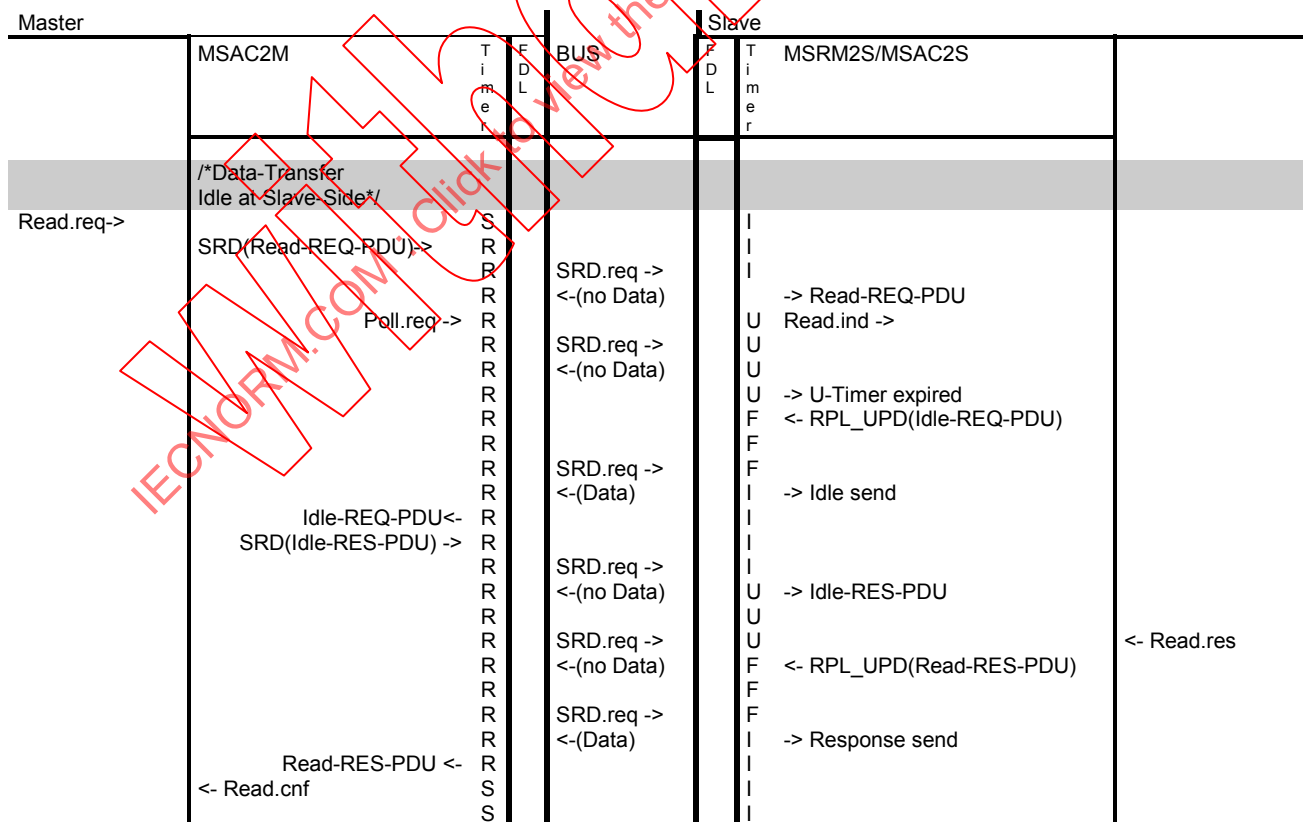


Figure 129 - Idle at Slave-Side on MS2

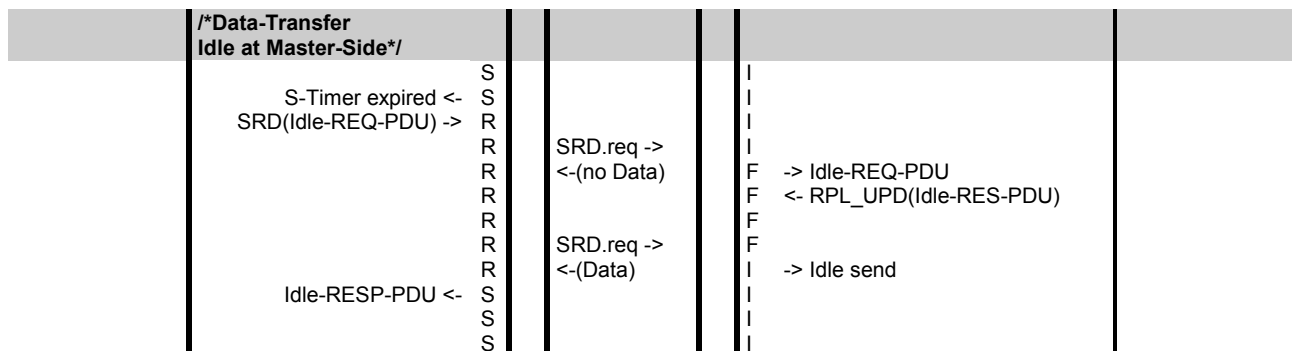


Figure 130 - Idle at Master-Side on MS2

Phase a) Connection monitoring during pending MS2 service response or running Master-Idle:

The DP-Master starts the R-Timer (receive timer) when the MS2 service request is transferred to the local DLL. The R-Timer monitors the remote MSAC2S and the local DLL.

The R-Timer is stopped if a response is received or restarted if a Slave-Idle request is received. The connection monitoring aborts the connection if a negative DL confirmation appears or the R-Timer expires.

The slave starts the U-Timer (user response timer) when a confirmed service indication is passed to the User. The U-Timer is stopped when the User provides the service response. If the U-Timer expires a Slave-Idle request is sent to retrigger the monitoring R-Timer of the Master.

The DP-Slave starts the F-Timer (fetch response timer) when a service response or Slave-Idle request is passed to the local DLL. The F-Timer is stopped when the master has fetched the PDU. If the F-Timer expires the connection monitoring aborts the connection.

Phase b) Connection monitoring without pending service response:

The bus master starts the S-Timer (send timer) if a response PDU or a Idle-PDU is received. The S-Timer is stopped if a service request is provided by the User. If the S-Timer expires a Master-Idle request is sent to stop the monitoring I-Timer of the Slave.

During pending Master-Idle responses the connection monitoring is handled as described in Phase a).

The Slave starts the I-Timer (Indication timer) when the reply update buffer has been fetched by the Master. The I-Timer is stopped if a Master-Idle request or a service indication is received. If the I-Timer expires the connection monitoring aborts the connection.

The Master monitors the transmission of the Abort-REQ-PDU. The Master starts the S-Timer when the Abort.req is transferred to the local DLL. The S-Timer is stopped when the Abort.req is sent. If the S-Timer expires the connection is closed and the User is informed with a MSAC2M_Closed.ind.

The Slave monitors the fetching of the Abort-REQ-PDU by the Master. The Slave starts the F-Timer when the Abort.req is transferred to the local DL. The F-Timer is stopped if the Master has fetched the Abort-REQ-PDU. If the F-Timer expires the Server_SAP is deactivated and the connection is closed.

6.4 AP Context State Machine

There is no AP Context State Machine defined for type 3 Protocol.

6.5 FAL Service Protocol Machines (FSPMs)

6.5.1 FSPMS

6.5.1.1 Primitive Definitions

6.5.1.1.1 Primitives Exchanged between FSPMS and AP-Context

Table 368 - Primitives issued by AP-Context to FSPMS

Primitive Name	Source	Associated Parameters	Functions
DLL Init DP Slave.req	AP-Context	Bus Para	Refer to FAL Type 3 Service Definition IEC 61158-5
SInit MS0.req	AP-Context	AREP	
SInit MS1.req	AP-Context	AREP	
SInit MS2.req	AP-Context	AREP	
Reset DP Slave.req	AP-Context	(none)	
Abort.req	AP-Context	AREP, Subnet, Instance, Reason Code	
DP Slave Application Ready.req	AP-Context	AREP	
Check User Prm Result.req	AP-Context	AREP, Prm_OK	
Check Cfg Result.req	AP-Context	AREP, Cfg_OK, Input Data Len, Output Data Len	
Set Cfg.req	AP-Context	AREP, Cfg Data	
Set Slave Diag.req	AP-Context	AREP, Ext Diag Overflow, Ext Diag Flag, Ext Diag Data	
Set Input.req	AP-Context	AREP, Input Data	
Get Output.req	AP-Context	AREP	
Alarm Notification.req	AP-Context	AREP, Slot Number, Alarm Type, Seq Nr, Add Ack, Alarm Specifier, Alarm Data	
Initiate.rsp(+)	AP-Context	AREP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
Initiate.rsp(-)	AP-Context	AREP, Error Decode, Error Code 1 Error Code 2	
Read.rsp(+)	AP-Context	AREP, Length, Data	
Read.rsp(-)	AP-Context	AREP, Error Decode, Error Code 1 Error Code 2	
Write.rsp(+)	AP-Context	AREP, Length	
Write.rsp(-)	AP-Context	AREP, Error Decode, Error Code 1 Error Code 2	
Data Transport.rsp(+)	AP-Context	AREP, Length, Data	
Data Transport.rsp(-)	AP-Context	AREP, Error Decode, Error Code 1 Error Code 2	
Alarm Ack.rsp(+)	AP-Context	AREP, Slot Number, Alarm Type, Seq Nr	

Primitive Name	Source	Associated Parameters	Functions
Alarm Ack.rsp(-)	AP-Context	AREP, Slot Number, Alarm Type, Seq Nr	
Load ARL DP Slave.req	AP-Context	Min Send Timeout, List of Resource Manager Entries, List of S_SAP_indices, List of ARL Entries	
Get ARL DP Slave.req	AP-Context	(none)	
Load CRL DP Slave.req	AP-Context	List of CRL Entries	
Get CRL DP Slave.req	AP-Context	(none)	

Table 369 - Primitives issued by FSPMS to AP-Context

Primitive Name	Source	Associated Parameters	Functions
SInit DLL.cnf	FSPMS	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
SInit MS0.cnf	FSPMS	AREP	
SInit MS1.cnf	FSPMS	AREP	
SInit MS2.cnf	FSPMS	AREP	
Reset.cnf	FSPMS	(none)	
Check User Prm Result.cnf(+)	FSPMS	AREP	
Check User Prm Result.cnf(-)	FSPMS	AREP, Status	
Check Cfg Result.cnf(+)	FSPMS	AREP	
Check Cfg Result.cnf(-)	FSPMS	AREP, Status	
Set Cfg.cnf(+)	FSPMS	AREP	
Set Cfg.cnf(-)	FSPMS	AREP	
Set Slave Diag.cnf	FSPMS	AREP	
Set Input.cnf(+)	FSPMS	AREP	
Set Input.cnf(-)	FSPMS	AREP	
Get Output.cnf(+)	FSPMS	AREP, Output Data, Clear Flag, New Flag	
Get Output.cnf(-)	FSPMS	AREP	
Started.ind	FSPMS	AREP, Actual Enabled Alarms, Alarm Sequence, Alarm Limit	
Stopped.ind	FSPMS	AREP	
Abort.ind	FSPMS	AREP, Locally Generated, Subnet, Instance, Reason Code, Additional Detail	
Fault.ind	FSPMS	AREP	
Alarm Reject.ind	FSPMS	AREP, Slot Number, Alarm Type, Seq Nr	
Set Slave Add.ind	FSPMS	AREP, New Slave Add, Ident Number, No Add Chg, Rem Slave Data	
Check User Prm.ind	FSPMS	AREP, User Prm Data	
Check Cfg.ind	FSPMS	AREP, Cfg Data	

Primitive Name	Source	Associated Parameters	Functions
New Output.ind	FSPMS	AREP, Output Data, Clear Flag	
Global Control.ind	FSPMS	AREP, Clear Command, Sync Command, Freeze Command, Group Select	
Initiate.ind	FSPMS	AREP, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
Read.ind	FSPMS	AREP, Slot Number, Index, Length	
Write.ind	FSPMS	AREP, Slot Number, Index, Length, Data	
Data Transport.ind	FSPMS	AREP, Slot Number, Index, Length, Data	
Alarm Ack.ind	FSPMS	AREP, Slot Number, Alarm Type, Seq Nr	
Load ARL DP Slave.cnf(+)	FSPMS	(none)	
Load ARL DP Slave.cnf(-)	FSPMS	Status	
Get ARL DP Slave.cnf(+)	FSPMS	Min Send Timeout, List of Resource Manager Entries, List of S_SAP_indices, List of ARL Entries	
Get ARL DP Slave.cnf(-)	FSPMS	Status	
Load CRL DP Slave.cnf(+)	FSPMS	(none)	
Load CRL DP Slave.cnf(-)	FSPMS	Status	
Get CRL DP Slave.cnf(+)	FSPMS	List of CRL Entries	
Get CRL DP Slave.cnf(-)	FSPMS	Status	

6.5.1.1.2 Parameters of FSPMS Primitives

The parameters used with the primitives exchanged between the FSPMS and the AP-Context are described in FAL Type 3 Service Definition IEC 61158-5.

6.5.1.2 State Machine Description

This Machine is used to co-ordinate the Interactions between AP-Context and DMPMS, MSCY1S, MSAC1S, MSRM2S and MSAC2S. As Alarms requires the support of several State Machines the alarms are handled by this machine.

With the SInit_MS1.req, the local variables of the State Machines are set to the start-up values and MSAC1 State Machines is initialised.

When the MSCY1S enters the Data-Exchange-State (This is signalled by a start.ind) alarm messages can be generated by the User.

The parallel processing of alarm messages is limited by:

the number of Alarm_Classes (Alarm_Sequence=False)

the number of Alarms accepted by the Master (Alarm_Sequence=True)

the number of Alarms handled by this Slave (Alarm_Sequence=True).

Alarms may be processed only if the appropriate Alarm_Class is enabled by the DP-Master. An Alarm issued by an Alarm_Notification will be transmitted via a diagnosis (MSCY1S) to a DP-Master (Class 1). The Acknowledgement will be received by a Alarm Ack service processed by MSAC1S.

The State Machine needs local variables which are described below. Furthermore the machine uses functions and macros which are listed in subclause 6.5.1.4.

Local variables of the MSAL1S

Alarm_Sequence

(Boolean)

This variable indicates whether

only one alarm of a specific Alarm_Class can be active at one time (Alarm_Sequence=False) or several alarms (2 to 32) of any type can be active at one time (Alarm_Sequence=True).

Initial_Alarm_Sequence

(Boolean)

This variable stores the basic ability of the Slave.

Outstanding_Alarm_TABLE

(Array 0...7, 0...31 of Alarm_Status, Alarm_Type)

Alarm_Status: pending, not_pending

Alarm_Type: 1 to 6, 32 to 126

The Alarm_State_Table is a two dimensional array with 7 * 32 elements. Each element within the array stores information about the actual state of any alarm. The array is addressed with the Alarm_Class calculated from the Alarm_Type and the Seq_Nr of the alarm.

Alarm_Max

(Unsigned8)

This variable indicates the maximum number of parallel alarms of this Slave.

Range: 7..32

Alarm_Limit

(Unsigned8)

This variable contains the maximum number of Alarms allowed by the actual Master-Slave-connection.

Range: 7..32

Alarm_DeCode

(Array 0...7 of Unsigned8)

Decoding the number of parallel alarms:

Alarm_DeCode[0]= 1
Alarm_DeCode[1]= 2
Alarm_DeCode[2]= 4
Alarm_DeCode[3]= 8
Alarm_DeCode[4]=12
Alarm_DeCode[5]=16
Alarm_DeCode[6]=24
Alarm_DeCode[7]=32

Alarm_Count

(Unsigned8)

This counter contains the number of alarms (0..32) which have actually been sent by the DP-Slave. The counter is only used in the sequence mode.

Range: 0 to Alarm_Limit

Req_Alarm_FIFO

The Req_Alarm_FIFO is used to store the Alarms which are still to send. Each element consists of an Alarm-PDU. The FIFO must be able to hold up to 32 respectively Alarm_Max entries.

Actual_Enabled_Alarms

(Array of Bits)

This variable indicates the type of alarms which are actually supported by the Master.

- Bit 0 = reserved
- Bit 1 = reserved
- Bit 2 = Update_Alarm
- Bit 3 = Status_Alarm
- Bit 4 = Manufacturer_Specific_Alarm
- Bit 5 = Diagnostic_Alarm
- Bit 6 = Process_Alarm
- Bit 7 = Pull_Plug_Alarm

Alarms_Supported

(Array of Bits)

This variable indicates the type of alarms which are supported by the Slave.

- Bit 0 = reserved
- Bit 1 = reserved
- Bit 2 = Update_Alarm
- Bit 3 = Status_Alarm
- Bit 4 = Manufacturer_Specific_Alarm
- Bit 5 = Diagnostic_Alarm
- Bit 6 = Process_Alarm
- Bit 7 = Pull_Plug_Alarm

Local_Diag_Buffer

(Octet-String)

This local variable is used to store the Diag-Data except the first 6 Bytes and the Alarm- and Status-PDUs.

L_Ext_Diag_Flag

(Bit)

This local variable is used to store the Ext_Diag_Flag.

Stored_Diag

(Boolean)

Flag which indicates, that new Diag_Data are stored in the Diag_Buffer while waiting for transmission of a previous Diagnosis.

Diag_Buffer

(Octet-String)

Buffer for storing a Slave-Diagnostic(Identification-Related-Diagnosis, Channel-Related-Diagnosis, Revision_Number)

Index

(Structure of Alarm_Class, Seq_Nr)

This index is used for addressing outstanding Alarm_Table.

6.5.1.3 FSPMS State Table

The following definition for a state set is used in the state table below.

t state = W-START or W-DIA-UPD or W-FETCHED

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	Load ARL DP Slave.req (Min Send Timeout, List of Resource Manager Entries, List of S_SAP_indices, List of ARL Entries) /valid parameters =>Load ARL DP Slave.cnf(+)	POWER-ON
2	POWER-ON	Load ARL DP Slave.req (Min Send Timeout, List of Resource Manager Entries, List of S_SAP_indices, List of ARL Entries) /invalid parameters =>Status := NO Load ARL DP Slave.cnf(-) (Status)	POWER-ON
3	POWER-ON	Get ARL DP Slave.req /ARL loaded =>Get ARL DP Slave.cnf(+) (Min Send Timeout, List of Resource Manager Entries, List of S_SAP_indices, List of ARL Entries)	POWER-ON
4	POWER-ON	Get ARL DP Slave.req /ARL not loaded =>Status := NO Get ARL DP Slave.cnf(-) (Status)	POWER-ON
5	POWER-ON	Load CRL DP Slave.req (List of CRL Entries) /valid parameters =>Load CRL DP Slave.cnf(+)	POWER-ON
6	POWER-ON	Load CRL DP Slave.req (List of CRL Entries) /invalid parameters =>Status := NO Load CRL DP Slave.cnf(-) (Status)	POWER-ON
7	POWER-ON	Get CRL DP Slave.req /CRL loaded =>Get CRL DP Slave.cnf(+) (List of CRL Entries)	POWER-ON
8	POWER-ON	Get CRL DP Slave.req /CRL not loaded =>Status := NO Get CRL DP Slave.cnf(-) (Status)	POWER-ON
9	POWER-ON	FSPMS_SInit_MS1.req (AREP) /Alarm_Mode_Slave <> 0 =>Alarm_Sequence := True Initial_Alarm_Sequence := Alarm_Sequence Alarm_Max := Alarm_Decode[Alarm_Mode_Slave] Alarm_Max := max(Alarm_Max, 7) Local_Diag_Buffer := Default_Ext_Diag_Data L_Ext_Diag_Flag := Default_Ext_Diag FILL (Outstanding_Alarm_TABLE.Alarm_Status, not_pending) Stored_Diag:=False Reset_Req_Alarm_FIFO() MSAC1_SInit_MS1.req	W-START-C1
10	POWER-ON	FSPMS_SInit_MS1.req (AREP) /Alarm_Mode_Slave = 0 =>Alarm_Sequence := False Initial_Alarm_Sequence := Alarm_Sequence Alarm_Max := 7 Local_Diag_Buffer := Default_Ext_Diag_Data L_Ext_Diag_Flag := Default_Ext_Diag FILL(Outstanding_Alarm_TABLE.Alarm_Status, not_pending) Stored_Diag:=False Reset_Req_Alarm_FIFO() FSPMS_User_Reset:=False MSAC1_SInit_MS1.req	W-START-C1
11	W-START-C1	MSAC1S_Sinit_MS1.cnf =>FSPMS_Sinit_MS1.cnf(AREP)	W-START
12	W-START	FSPMS_Abort.req(AREP) /AREP.AR_Type=MS0 =>MSCY1S_Abort.req	W-START
13	W-START	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) =>FSPMS_Alarm_Notification.cnf(-)(AREP, Alarm_Type, Slot_Number, Seq_Nr, Status=No_Start)	W-START

#	Current State	Event /Condition =>Actions taken&&Output	Next State
14	W-START	FSPMS_Set_Slave_Diag.req(AREP,Ext_Diag_Flag, Ext_Diag) =>L_Ext_Diag_Flag := Ext_Diag_Flag L_Ext_Diag_Overflow := Ext_Diag_Overflow Local_Diag_Buffer := Ext_Diag MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) FSPMS_Set_Slave_Diag.cnf(AREP)	W-START
15	W-START	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) =>ignore	W-START
16	W-START	MSCY1S_Set_Slave_Diag.cnf(-) =>ignore	W-START
17	W-START	MSCY1S_Set_Slave_Diag.cnf(+) =>ignore	W-START
20	W-START	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) /(Enabled Alarms & Alarms_Supported) = 0 =>ignore	W-START
18	W-START	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) / ((Enabled Alarms & Alarms_Supported) <> 0) && Alarm_Mode_Master <> 0 && Initial_Alarm_Sequence = TRUE && Diag_Flag=TRUE =>Actual_Enabled_Alarms := Enabled_Alarms Alarm_Limit := min(Alarm_Max, Alarm_DeCode[Alarm_Mode_Master]) Alarm_Count := 0 FSPMS DP Slave Started.ind(AREP,Actual_Enabled_Alarms, Alarm_Sequence, Alarm_Limit)	W-FETCH ED
19	W-START	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) /((Enabled Alarms & Alarms_Supported) <> 0) && Alarm_Mode_Master <> 0 && Initial_Alarm_Sequence = TRUE && Diag_Flag=FALSE =>Actual_Enabled_Alarms := Enabled_Alarms Alarm_Limit := min(Alarm_Max, Alarm_DeCode[Alarm_Mode_Master]) Alarm_Count := 0 FSPMS DP Slave Started.ind(AREP,Actual_Enabled_Alarms, Alarm_Sequence, Alarm_Limit)	W-DIA-UPD
20	W-START	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) /((Enabled Alarms & Alarms_Supported) <> 0) && Alarm_Mode_Master <> 0 && Initial_Alarm_Sequence =FALSE =>MSCY1S_Abort.req	W-START
21	W-START	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) /((Enabled Alarms & Alarms_Supported) <> 0) && Alarm_Mode_Master = 0 && Diag_Flag=TRUE =>Alarm_Sequence := False Actual_Enabled_Alarms := Enabled_Alarms FSPMS DP Slave Started.ind(AREP,Actual_Enabled_Alarms, Alarm_Sequence, Alarm_Limit)	W-FETCH ED
22	W-START	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) /((Enabled Alarms & Alarms_Supported) <> 0) && Alarm_Mode_Master = 0 && Diag_Flag=FALSE =>Alarm_Sequence := False Actual_Enabled_Alarms := Enabled_Alarms FSPMS DP Slave Started.ind(AREP,Actual_Enabled_Alarms, Alarm_Sequence, Alarm_Limit)	W-DIA-UPD
23	W-START	MSCY1S_Stop.ind =>ignore	W-START
24	W-DIA-UPD	FSPMS_Abort.req(AREP) /AREP.AR_Type=MS0 =>Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) MSCY1S_Abort.req	RESET-P-LEAVE
25	W-DIA-UPD	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=FALSE =>FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number, Seq_Nr, Status=Not_Enabled)	W-DIA-UPD
26	W-DIA-UPD	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=TRUE && Alarm_Count=Alarm_Limit =>FSPMS_Alarm_Notification.cnf(-)(AREP, Alarm_Type, Slot_Number, Seq_Nr, Status=Limit_Expired)	W-DIA-UPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
27	W-DIA-UPD	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=TRUE && Alarm_Count<Alarm_Limit && Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]=not_pending =>Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]:=pending Outstanding_Alarm_TABLE.Alarm_Type[Acls(Alarm_Type),Seq_Nr]:=Alarm_Type Alarm_Count:=Alarm_Count+1 Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer, Alarm(Slot_Number,Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data)) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) FSPMS_Alarm_Notification.cnf(+)(AREP,Alarm_Type, Slot_Number, Seq_Nr)	W-FETCH ED
28	W-DIA-UPD	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=TRUE && Alarm_Count<Alarm_Limit && Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]=pending =>FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number, Seq_Nr, Status=Sequence_Nr_Pending)	W-DIA-UPD
29	W-DIA-UPD	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=FALSE && Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]=pending =>FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number, Seq_Nr, Status=Sequence_Nr_Pending)	W-DIA-UPD
30	W-DIA-UPD	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=FALSE && Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]=not_pending =>Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]:=pending Outstanding_Alarm_TABLE.Alarm_Type[Acls(Alarm_Type),Seq_Nr]:=Alarm_Type Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer, Alarm(Slot_Number,Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data)) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) FSPMS_Alarm_Notification.cnf(+)(AREP,Alarm_Type, Slot_Number, Seq_Nr)	W-FETCH ED
31	W-DIA-UPD	FSPMS_Set_Slave_Diag.req(AREP,Ext_Diag_Flag, Ext_Diag) =>L_Ext_Diag_Flag := Ext_Diag_Flag L_Ext_Diag_Overflow := Ext_Diag_Overflow Local_Diag_Buffer := Ext_Diag MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) FSPMS_Set_Slave_Diag.cnf(AREP)	W-DIA-UPD
32	W-DIA-UPD	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) /Alarm_Sequence=TRUE && Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr]=pending =>Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr] := not_pending Alarm_Count:=Alarm_Count-1 FSPMS_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr)	W-DIA-UPD
33	W-DIA-UPD	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) /Alarm_Sequence=FALSE && Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr] =pending =>Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr] := not_pending FSPMS_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr)	W-DIA-UPD
34	W-DIA-UPD	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) /Outstanding_Alarm_TABLE.Alarm_Status[Acls(Alarm_Type),Seq_Nr] =not_pending =>Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) MSCY1S_Abort.req FSPMS_Abort.ind(AREP)	RESET-P-LEAVE
35	W-DIA-UPD	MSCY1S_Set_Slave_Diag.cnf(-) =>R_Cnt:=0 FSPMS_DP_Slave_Fault.ind DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W-RESET
36	W-DIA-UPD	MSCY1S_Set_Slave_Diag.cnf(+) =>Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) MSCY1S_Abort.req FSPMS_Abort.ind(AREP)	RESET-P-LEAVE

#	Current State	Event /Condition =>Actions taken&&Output	Next State
37	W-DIA-UPD	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) =>MSCY1S_Abort.req	W-START
38	W-DIA-UPD	MSCY1S_Stop.ind =>Alarm_Count:=0 Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending)	RESET-PENDING
39	RESET-PENDING	/Index.Seq_Nr <> 255 =>Outstanding_Alarm_TABLE.Alarm_Status[Index] := not_pending Alarm_Type := Outstanding_Alarm_TABLE.Alarm_Type[AcIs(Alarm_Type),Seq_Nr] Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number=0, Index.Seq_Nr, Status=MSAL1S_Stopped)	RESET-PENDING
40	RESET-PENDING	/Index.Seq_Nr = 255 =>Reset_Req_Alarm_FIFO() FSPMS DP Slave Stopped.ind(AREP)	W-START
41	RESET-P-LEAVE	/Index.Seq_Nr <> 255 =>Outstanding_Alarm_TABLE.Alarm_Status[Index] := not_pending Alarm_Type:= Outstanding_Alarm_TABLE.Alarm_Type[AcIs(Alarm_Type),Seq_Nr] Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number=0, Index.Seq_Nr, Status=Stopped)	RESET-P-LEAVE
42	RESET-P-LEAVE	/Index.Seq_Nr = 255 =>Reset_Req_Alarm_FIFO()	W-START
43	W-FETCHED	FSPMS_Abort.req(AREP) /AREP.AR_Type=MS0 && Alarm_Sequence=TRUE && Alarm_Count=0 =>Stored_Diag:=False Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) MSCY1S_Abort.req	W-START
44	W-FETCHED	FSPMS_Abort.req(AREP) /AREP.AR_Type=MS0 && Alarm_Sequence=FALSE Alarm_Count<>0 =>Stored_Diag:=False Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer) Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) MSCY1S_Abort.req	RESET-P-LEAVE
45	W-FETCHED	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=FALSE =>FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number, Seq_Nr, Status=Not_Enabled)	W-FETCHED
46	W-FETCHED	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=TRUE && Alarm_Count=Alarm_Limit =>FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number, Seq_Nr, Status=Limit_Expired)	W-FETCHED
47	W-FETCHED	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=TRUE && Alarm_Count<Alarm_Limit && Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]=not_pending =>Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]:=pending Alarm_Count:=Alarm_Count+1 Store_to_Req_Alarm_FIFO(Alarm(Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data)) FSPMS_Alarm_Notification.cnf(+)(AREP,Alarm_Type, Slot_Number, Seq_Nr)	W-FETCHED
48	W-FETCHED	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=TRUE && Alarm_Count<Alarm_Limit && Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]= pending =>FSPMS_Alarm_Notification.cnf(-)(AREP,Alarm_Type, Slot_Number, Seq_Nr, Status=Sequence_Nr_Pending)	W-FETCHED

#	Current State	Event /Condition =>Actions taken&&Output	Next State
49	W-FETCHED	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=FALSE && Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]= pending =>FSPMS_Alarm_Notification.cnf(-)(AREP, Alarm_Type, Slot_Number, Seq_Nr, Status=Sequence_Nr_Pending)	W-FETCHED
50	W-FETCHED	FSPMS_Alarm_Notification.req(AREP, Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data) /ALARM_ENABLED=TRUE && Alarm_Sequence=FALSE && Outstanding_Alarm_TABLE.Alarm_Status[AcIsL(Alarm_Type),Seq_Nr]= not_pending =>Outstanding_Alarm_TABLE.Alarm_Status[AcIsL(Alarm_Type),Seq_Nr] := pending Outstanding_Alarm_TABLE.Alarm_Type[AcIs(Alarm_Type),Seq_Nr] := Alarm_Type Store_to_Req_Alarm_FIFO(Alarm(Slot_Number, Alarm_Type, Seq_Nr, Add_Ack, Alarm_Specifier, Alarm_Data)) FSPMS_Alarm_Notification.cnf(+)(AREP,Alarm_Type, Slot_Number, Seq_Nr)	W-FETCHED
51	W-FETCHED	FSPMS_Set_Slave_Diag.req(AREP,Ext_Diag_Flag, Ext_Diag) =>L_Ext_Diag_Flag := Ext_Diag_Flag L_Ext_Diag_Overflow := Ext_Diag_Overflow Local_Diag_Buffer := Ext_Diag FSPMS_Set_Slave_Diag.cnf(AREP)	W-FETCHED
52	W-FETCHED	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) /Alarm_Sequence=TRUE && Outstanding_Alarm_TABLE.Alarm_Status[AcIs (Alarm_Type),Seq_Nr]=pending =>Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]=not_pending Alarm_Count:=Alarm_Count-1 FSPMS_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr)	W-FETCHED
53	W-FETCHED	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) /Alarm_Sequence=FALSE && Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]=pending =>Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]=not_pending FSPMS_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr)	W-FETCHED
54	W-FETCHED	MSAC1S_Alarm_Ack.ind(Slot_Number, Alarm_Type, Seq_Nr) /Outstanding_Alarm_TABLE.Alarm_Status[AcIs(Alarm_Type),Seq_Nr]=not_pending =>Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) MSCY1S_Abort.req FSPMS_Abort.ind(AREP)	RESET-P-LEAVE
55	W-FETCHED	MSCY1S_Set_Slave_Diag.cnf(-) =>R_Cnt:=0 FSPMS_DP_Slave_Fault.ind DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W-RESET
56	W-FETCHED	MSCY1S_Set_Slave_Diag.cnf(+)	CHK-TRANSMIT
57	W-FETCHED	MSCY1S_Start.ind(Enabled_Alarms, Alarm_Mode_Master, Diag_Flag) =>MSCY1S_Abort.req	W-START
58	W-FETCHED	MSCY1S_Stop.ind /Alarm_Sequence=TRUE && Alarm_Count=0 =>Stored_Diag:=False Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag) FSPMS_Stopped.ind(AREP)	W-START
59	W-FETCHED	MSCY1S_Stop.ind /Alarm_Sequence=FALSE Alarm_Count<>0 =>Stored_Diag:=False Alarm_Count:=0 Index := Search_In_Outstanding_Alarm_TABLE(Alarm_Status:=pending) Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag)	RESET-PENDING
60	CHK-TRANSMIT	/Req_Alarm_FIFO_state=empty &&Stored_Diag = FALSE	W-DIA-UPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
61	CHK- TRANSM IT	/Req_Alarm_FIFO_state=empty && Stored_Diag = TRUE =>Stored_Diag:=False Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag)	W- FETCH ED
62	CHK- TRANSM IT	/Req_Alarm_FIFO_state<>empty && Stored_Diag = FALSE =>Alarm:=Load_from_Req_Alarm_FIFO() Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer, Alarm) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag)	W- FETCH ED
63	CHK- TRANSM IT	/Req_Alarm_FIFO_state<>empty && Stored_Diag = TRUE =>Alarm:=Load_from_Req_Alarm_FIFO() Stored_Diag:=False Ext_Diag_Flag:=L_Ext_Diag_Flag Ext_Diag_Overflow:=L_Ext_Diag_Overflow Ext_Diag:=(Local_Diag_Buffer, Alarm) MSCY1S_Set_Slave_Diag.req(Ext_Diag_Flag, Ext_Diag_Overflow, Ext_Diag)	W- FETCH ED
64	W- RESET	DMPMS_Reset.cnf /R_Cnt<3 =>R_Cnt:=R_Cnt+1	W- RESET
65	W- RESET	MSCY1S_Reset.cnf /R_Cnt<3 =>R_Cnt:=R_Cnt+1	W- RESET
66	W- RESET	MSAC1S_Reset.cnf /R_Cnt<3 =>R_Cnt:=R_Cnt+1	W- RESET
67	W- RESET	MSRM2S_Reset.cnf /R_Cnt<3 =>R_Cnt:=R_Cnt+1	W- RESET
68	W- RESET	DMPMS_Reset.cnf /R_Cnt>=3 =>if (FSPMS_User_Reset=True) FSPMS_Reset DP Slave .cnf endif	POWER -ON
69	W- RESET	MSCY1S_Reset.cnf /R_Cnt>=3 =>if (FSPMS_User_Reset=True) FSPMS_Reset DP Slave .cnf endif	POWER -ON
70	W- RESET	MSAC1S_Reset.cnf /R_Cnt>=3 =>if (FSPMS_User_Reset=True) FSPMS_Reset DP Slave .cnf endif	POWER -ON
71	W- RESET	MSRM2S_Reset.cnf /R_Cnt>=3 =>if (FSPMS_User_Reset=True) FSPMS_Reset DP Slave .cnf endif	POWER -ON
72	any state	DMPMS_Fault.ind =>R_Cnt:=0 FSPMS DP Slave Fault.ind DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W- RESET

#	Current State	Event /Condition =>Actions taken&&Output	Next State
73	any state	MSAC1S_Fault.ind =>R_Cnt:=0 FSPMS_DP_Slave_Fault.ind DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W- RESET
74	any state	MSCY1S_Fault.ind =>R_Cnt:=0 FSPMS_DP_Slave_Fault.ind DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W- RESET
75	any state	MSRM2S_Fault.ind =>R_Cnt:=0 FSPMS_DP_Slave_Fault.ind DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W- RESET
76	any state	FSPMS_Reset_DP_Slave.req =>R_Cnt:=0 FSPMS_User_Reset:=True DMPMS_Reset.req MSCY1S_Reset.req MSAC1S_Reset.req MSRM2S_Reset.req	W- RESET
77	t-state	FSPMS_DLL_Init_DP_Slave.req (Bus_Para) => DMPMS_Sinit_DLL.req (Bus_Para)	SAME
78	t-state	FSPMS_Sinit_MS0.req (AREP) => MSCY1S_Sinit_MS0.req ()	SAME
79	t-state	FSPMS_Sinit_MS2.req () => MSRM2S_Sinit_MS2.req ()	SAME
80	t-state	FSPMS_Abort.req (AREP, Subnet, Instance, Reason Code) /AREP.AR_Type=MS2 => MSAC2S_Abort.req (Subnet, Instance, Reason Code)	SAME
81	t-state	FSPMS_DP_Slave_Application_Ready.req (AREP) /AREP.AR_Type=MS0 => MSCY1S_Application_Ready.req ()	SAME
82	t-state	FSPMS_Check_User_Prm_Result.req (AREP, Prm_OK) /AREP.AR_Type=MS0 => MSCY1S_Check_User_Prm_Result.req (Prm_OK)	SAME
83	t-state	FSPMS_Check_Cfg_Result.req (AREP, Cfg_OK, Input Data Len, Output Data Len) /AREP.AR_Type=MS0 => MSCY1S_Check_Cfg_Result.req (Cfg_OK, Input Data Len, Output Data Len)	SAME
84	t-state	FSPMS_Set_Cfg.req (AREP, Cfg Data) /AREP.AR_Type=MS0 => MSCY1S_Set_Cfg.req (Cfg Data)	SAME
85	t-state	FSPMS_Set_Input.req (AREP, Input Data) /AREP.AR_Type=MS0 => MSCY1S_Set_Input.req (Input Data)	SAME
86	t-state	FSPMS_Get_Output.req (AREP) /AREP.AR_Type=MS0 => MSCY1S_Get_Output.req ()	SAME
87	t-state	FSPMS_Initiate.rsp(+) (AREP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param) /AREP.AR_Type=MS1 => MSAC2S_Initiate.rsp(+) (Res SAP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param)	SAME
88	t-state	FSPMS_Initiate.rsp(-) (AREP, Error Decode, Error Code 1 Error Code 2) /AREP.AR_Type=MS2 => MSAC2S_Initiate.rsp(-) (Res SAP, Error Decode, Error Code 1 Error Code 2)	SAME
89	t-state	FSPMS_Read.rsp(+) (AREP, Length, Data) /AREP.AR_Type=MS1 => MSAC1S_Read.rsp(+) (Length, Data)	SAME
90	t-state	FSPMS_Read.rsp(+) (AREP, Length, Data) /AREP.AR_Type=MS2 => MSAC2S_Read.rsp(+) (Length, Data)	SAME

#	Current State	Event /Condition =>Actions taken&&Output	Next State
91	t-state	FSPMS_Read.rsp(-) (AREP, Error Decode, Error Code 1 Error Code 2) /AREP.AR_Type=MS1 => MSAC1S_Read.rsp(-) (Error Decode, Error Code 1 Error Code 2)	SAME
92	t-state	FSPMS_Read.rsp(-) (AREP, Error Decode, Error Code 1 Error Code 2) /AREP.AR_Type=MS2 => MSAC2S_Read.rsp(-) (Error Decode, Error Code 1 Error Code 2)	SAME
93	t-state	FSPMS_Write.rsp(+) (AREP, Length) /AREP.AR_Type=MS1 => MSAC1S_Write.rsp(+) (Length)	SAME
94	t-state	FSPMS_Write.rsp(+) (AREP, Length) /AREP.AR_Type=MS2 => MSAC2S_Write.rsp(+) (Length)	SAME
95	t-state	FSPMS_Write.rsp(-) (AREP, Error Decode, Error Code 1 Error Code 2) /AREP.AR_Type=MS1 => MSAC1S_Write.rsp(-) (Error Decode, Error Code 1 Error Code 2)	SAME
96	t-state	FSPMS_Write.rsp(-) (AREP, Error Decode, Error Code 1 Error Code 2) /AREP.AR_Type=MS2 => MSAC2S_Write.rsp(-) (Error Decode, Error Code 1 Error Code 2)	SAME
97	t-state	FSPMS_Data Transport.rsp(+) (AREP, Length, Data) /AREP.AR_Type=MS2 => MSAC2S_Data Transport.rsp(+) (Res SAP, Length, Data)	SAME
98	t-state	FSPMS_Data Transport.rsp(-) (AREP, Error Decode, Error Code 1 Error Code 2) /AREP.AR_Type=MS2 => MSAC2S_Data Transport.rsp(-) (Res SAP, Error Decode, Error Code 1 Error Code 2)	SAME
99	t-state	FSPMS_Alarm Ack.rsp(+) (AREP, Slot Number, Alarm Type, Seq Nr) /AREP.AR_Type=MS1 => MSAC1S_Alarm Ack.rsp(+) (Slot Number, Alarm Type, Seq Nr)	SAME
100	t-state	FSPMS_Alarm Ack.rsp(-) (AREP, Slot Number, Alarm Type, Seq Nr) /AREP.AR_Type=MS1 => MSAC1S_Alarm Ack.rsp(-) ()	SAME
101	t-state	DMPMS_SInit DLL.cnf () =>FSPMS_DLL Init DP Slave.cnf ()	SAME
102	t-state	MSCY1S_SInit MS0.cnf () =>AREP=MS0-AR FSPMS_SInit MS0.cnf (AREP)	SAME
103	t-state	MSRM2S_SInit MS2.cnf () =>FSPMS_SInit MS2.cnf ()	SAME
104	t-state	MSCY1S_Check User Prm Result.cnf(+) () =>AREP=MS0-AR FSPMS_Check User Prm Result.cnf(+) (AREP)	SAME
105	t-state	MSCY1S_Check User Prm Result.cnf(-) (Status) =>AREP=MS0-AR FSPMS_Check User Prm Result.cnf(-) (AREP, Status)	SAME
106	t-state	MSCY1S_Check Cfg Result.cnf(+) () =>AREP=MS0-AR FSPMS_Check Cfg Result.cnf(+) (AREP)	SAME
107	t-state	MSCY1S_Check Cfg Result.cnf(-) (Status) =>AREP=MS0-AR FSPMS_Check Cfg Result.cnf(-) (AREP, Status)	SAME
108	t-state	MSCY1S_Set Cfg.cnf(+) () =>AREP=MS0-AR FSPMS_Set Cfg.cnf(+) (AREP)	SAME
109	t-state	MSCY1S_Set Cfg.cnf(-) () =>AREP=MS0-AR FSPMS_Set Cfg.cnf(-) (AREP)	SAME
110	t-state	MSCY1S_Set Input.cnf(+) () =>AREP=MS0-AR FSPMS_Set Input.cnf(+) (AREP)	SAME
111	t-state	MSCY1S_Set Input.cnf(-) () =>AREP=MS0-AR FSPMS_Set Input.cnf(-) (AREP)	SAME
112	t-state	MSCY1S_Get Output.cnf(+) (Output Data, Clear Flag, New Flag) =>AREP=MS0-AR FSPMS_Get Output.cnf(+) (AREP, Output Data, Clear Flag, New Flag)	SAME
113	t-state	MSCY1S_Get Output.cnf(-) () =>AREP=MS0-AR FSPMS_Get Output.cnf(-) (AREP)	SAME

#	Current State	Event /Condition =>Actions taken&&Output	Next State
114	t-state	MSCY1S_Set Slave Add.ind (New Slave Add, Ident Number, No Add Chg, Rem Slave Data) =>AREP=MS0-AR FSPMS_Set Slave Add.ind (AREP, New Slave Add, Ident Number, No Add Chg, Rem Slave Data)	SAME
115	t-state	MSCY1S_Check User Prm.ind (User Prm Data) =>AREP=MS0-AR FSPMS_Check User Prm.ind (AREP, User Prm Data)	SAME
116	t-state	MSCY1S_Check Cfg.ind (Check Cfg Mode, Cfg Data) =>AREP=MS0-AR FSPMS_Check Cfg.ind (AREP, Check Cfg Mode, Cfg Data)	SAME
117	t-state	MSCY1S_New Output.ind (Clear Flag) =>AREP=MS0-AR FSPMS_New Output.ind (AREP, Clear Flag)	SAME
118	t-state	MSCY1S_Global Control.ind (Clear Command, Sync Command, Freeze Command, Group Select) =>AREP=MS0-AR FSPMS_Global Control.ind (AREP, Clear Command, Sync Command, Freeze Command, Group Select)	SAME
119	t-state	MSAC2S_Initiate.ind (Res SAP, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param) /SetContext(Res_SAP, Service)=TRUE =>FSPMS_Initiate.ind (AREP, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param)	SAME
120	t-state	MSAC2S_Read.ind (Res SAP, Slot Number, Index, Length) /SetContext(Res_SAP, Service)=TRUE =>FSPMS_Read.ind (AREP, Slot Number, Index, Length)	SAME
121	t-state	MSAC1S_Read.ind (Res SAP, Slot Number, Index, Length) /SetContext(Res_SAP, Service)=TRUE =>FSPMS_Read.ind (AREP, Slot Number, Index, Length)	SAME
122	t-state	MSAC2S_Write.ind (Res SAP, Slot Number, Index, Length, Data) /SetContext(Res_SAP, Service)=TRUE =>FSPMS_Write.ind (AREP, Slot Number, Index, Length, Data)	SAME
123	t-state	MSAC1S_Write.ind (Res SAP, Slot Number, Index, Length, Data) /SetContext(Res_SAP, Service)=TRUE =>FSPMS_Write.ind (AREP, Slot Number, Index, Length, Data)	SAME
124	t-state	MSAC2S_Data Transport.ind (Res SAP, Slot Number, Index, Length, Data) /SetContext(Res_SAP, Service)=TRUE =>FSPMS_Data Transport.ind (AREP, Slot Number, Index, Length, Data)	SAME

6.5.1.4 Functions

Table 370 - Function SetContext()

Function Name	Description
SetContext(Rem_Add, Service)	This function checks if there exist an entry for the inputs Rem_Add and invoked Service in the ARL and related CRL of the DP-Slave. A) If there exist an entry than it returns TRUE and sets the local context according to the current CREP and AREP. B) Otherwise it returns FALSE.

Table 371 - Function FILL()

Function Name	Description
FILL(Outstanding_Alarm_TABLE.Alarm_Status, State)	This function sets each entry of the two dimensional Outstanding_Alarm_TABLE to the given initial value of State. Valid parameters for State are pending/not_pending. The function has no return value.

Table 372 - Function Search_In_Outstanding_Alarm_TABLE ()

Function Name	Description
Search_In_Outstanding_Alarm_TABLE (Alarm_Status)	This function scans each entry of the two dimensional Outstanding_Alarm_TABLE for a specific Alarm_Status. The return value of this function is an index that refers to an Alarm_Table entry that fulfills the Alarm_Status criteria pending/not_pending.

Table 373 - Function SReset_Req_Alarm_FIFO ()

Function Name	Description
SReset_Req_Alarm_FIFO()	This function resets the FIFO for alarms not yet sent. During this function all entries are cleared. The function has no return value.

Table 374 - Function Store_To_Req_Alarm_FIFO ()

Function Name	Description
Store_To_Req_Alarm_FIFO(Alarm)	By means of this function the Alarm is stored into the Req_Alarm_FIFO. The function has no return value.

Table 375 - Function Load_From_Req_Alarm_FIFO ()

Function Name	Description
Load_From_Req_Alarm_FIFO ()	By means of this function one Alarm is read from the Req_Alarm_FIFO. The return value is an Alarm.

Table 376 - Function Req_Alarm_FIFO_state ()

Function Name	Description
Req_Alarm_FIFO_state ()	This function checks the FIFO for alarms not yet sent. The function has the return values empty/not_empty.

Table 377 - Function Acls ()

Function Name	Description
Acls(Alarm_Type)	This function calculates the Alarm_Type related index as return value as follows: IF (Alarm_Type = 1) THEN return 0 IF (Alarm_Type = 2) THEN return 1 IF (Alarm_Type = 3) THEN return 2 IF (Alarm_Type = 4) THEN return 3 IF (Alarm_Type = 5) THEN return 4 IF (Alarm_Type = 6) THEN return 5 IF (Alarm_Type >= 32) && (Alarm_Type <= 126) THEN return

Macro

ALARM_ENABLED

```
(
  ((Alarm_Type=3 OR Alarm_Type=4) &&
  Actual_Enabled_Alarms[7]=TRUE)
  OR ((Alarm_Type=2) && Actual_Enabled_Alarms[6]=TRUE)
  OR ((Alarm_Type=1) && Actual_Enabled_Alarms[5]=TRUE)
  OR ((Alarm_Type>=32 && Alarm_Type<=126)
  && Actual_Enabled_Alarms[4]=TRUE)
  OR ((Alarm_Type=5) && Actual_Enabled_Alarms[3]=TRUE)
  OR ((Alarm_Type=6) && Actual_Enabled_Alarms[2]=TRUE)
)
```

6.5.2 FSPMM1

6.5.2.1 Primitive Definitions

6.5.2.1.1 Primitives Exchanged between AP-Context and FSPMM1

Table 378 - Primitives issued by AP-Context to FSPMM1

Primitive Name	Source	Associated Parameters	Functions
Init DP Master Cl1.req	AP-Context	Bus Para	Refer to FAL Type 3 Service Definition IEC 61158-5 and IEC 61158-3
Reset DP Master Cl1.req	AP-Context	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Load ARL DP Master Cl1.req	AP-Context	Alarm Max, List of ARL Entries	
Get ARL DP Master Cl1.req	AP-Context	(none)	
Load CRL DP Master Cl1.req	AP-Context	List of CRL Entries	
Get CRL DP Master Cl1.req	AP-Context	(none)	
Abort DP Master Cl1.req	AP-Context	AREP	
Mark DP Master Cl1.req	AP-Context	AREP	
Set Mode DP Master Cl1.req	AP-Context	AREP, USIF State	
Load Bus Par DP Master Cl1.req	AP-Context	Bus Para	
Delete SC DP Master Cl1.req	AP-Context	Address	
Read Value DP Master Cl1.req	AP-Context	Variable	
Get Slave Diag.req	AP-Context	AREP, CREP	
Set Output.req	AP-Context	AREP, CREP, Slot Number, Output Data	
Get Input.req	AP-Context	AREP, CREP, Slot Number	
Global Control.req	AP-Context	AREP, Sync Command, Freeze Command, Group Select	
Read.req	AP-Context	AREP, Slot Number, Index, Length	
Write.req	AP-Context	AREP, Slot Number, Index, Length, Data	

Primitive Name	Source	Associated Parameters	Functions
Alarm Ack.req(+)	AP-Context	AREP, Slot Number, Alarm Type, Seq Nr	
Alarm Ack.cnf(-)	FSPMM1	AREP Slot Number, Alarm Type, Seq Nr	
Get Master Diag.rsp(+)	AP-Context	AREP, Diagnosis Data	
Get Master Diag.rsp(-)	AP-Context	AREP, Status	
Start Seq.rsp(+)	AP-Context	AREP, Max Len Data Unit	
Start Seq.rsp(-)	AP-Context	AREP, Status	
Download.rsp(+)	AP-Context	AREP	
Download.rsp(-)	AP-Context	AREP, Status	
Upload.rsp(+)	AP-Context	AREP, Data	
Upload.rsp(-)	AP-Context	AREP, Status	
End Seq.rsp(+)	AP-Context	AREP	
End Seq.rsp(-)	AP-Context	AREP, Status	
Act Param.rsp(+)	AP-Context	AREP	
Act Param.rsp(-)	AP-Context	AREP, Status	
Load ARL DP Master Cl1.req	AP-Context	Alarm Max, List of ARL Entries	
Get ARL DP Master Cl1.req	AP-Context	(none)	
Load CRL DP Master Cl1.req	AP-Context	List of CRL Entries	
Get CRL DP Master Cl1.req	AP-Context	(none)	

Table 379 - Primitives issued by FSPMM1 to AP-Context

Primitive Name	Source	Associated Parameters	Functions
Init DP Master Cl1.cnf	FSPMM1	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset DP Master Cl1.cnf	FSPMM1	(none)	
Load ARL DP Master Cl1.cnf(+)	FSPMM1	(none)	
Load ARL DP Master Cl1.cnf(-)	FSPMM1	Status	
Get ARL DP Master Cl1.cnf(+)	FSPMM1	Alarm Max, List of ARL Entries	
Get ARL DP Master Cl1.cnf(-)	FSPMM1	Status	
Load CRL DP Master Cl1.cnf(+)	FSPMM1	(none)	
Load CRL DP Master Cl1.cnf(-)	FSPMM1	Status	
Get CRL DP Master Cl1.cnf(+)	FSPMM1	List of CRL Entries	
Get CRL DP Master Cl1.cnf(-)	FSPMM1	Status	
Mark DP Master Cl1.cnf(+)	FSPMM1	AREP, Dia	
Mark DP Master Cl1.cnf(-)	FSPMM1	AREP, Status	
Set Mode DP Master Cl1.cnf(+)	FSPMM1	AREP, Bus Accessible	
Set Mode DP Master Cl1.cnf(-)	FSPMM1	AREP, Bus Accessible	
Load Bus Par DP Master Cl1.cnf	FSPMM1	Status	

Primitive Name	Source	Associated Parameters	Functions
Delete SC DP Master Cl1.cnf	FSPMM1	Status	
Read Value DP Master Cl1.cnf	FSPMM1	Value, Status	
Get Slave Diag.cnf(+)	FSPMM1	AREP, CREP, Diag Data	
Get Slave Diag.cnf(-)	FSPMM1	AREP, CREP	
Set Output.cnf(+)	FSPMM1	AREP, CREP, Slot Number	
Set Output.cnf(-)	FSPMM1	AREP, CREP, Slot Number	
Get Input.cnf(+)	FSPMM1	AREP, CREP, Slot Number, Input Data	
Get Input.cnf(-)	FSPMM1	AREP, CREP, Slot Number	
Global Control.cnf(+)	FSPMM1	AREP	
Global Control.cnf(-)	FSPMM1	AREP, Status	
Read.cnf(+)	FSPMM1	AREP, Length, Data	
Read.cnf(-)	FSPMM1	AREP, Error Decode, Error Code 1 Error Code 2	
Write.cnf(+)	FSPMM1	AREP, Length	
Write.cnf(-)	FSPMM1	AREP, Error Decode, Error Code 1 Error Code 2	
Alarm Ack.cnf(+)	FSPMM1	AREP, Slot Number, Alarm Type, Seq Nr	
Alarm Ack.cnf(-)	FSPMM1	AREP, Slot Number, Alarm Type, Seq Nr	
Get Master Diag.ind	FSPMM1	AREP, MDiag Identifier	
Start Seq.ind	FSPMM1	AREP, Area Code, Timeout	
Download.ind	FSPMM1	AREP, Area Code, Add Offset, Data	
Upload.ind	FSPMM1	AREP, Area Code, Add Offset, Data Len	
End Seq.ind	FSPMM1	AREP	
Act Param.ind	FSPMM1	AREP, Area Code, Activate	
Act Para Brct.ind	FSPMM1	AREP, Area Code	
New Slave Diag.ind	FSPMM1	AREP, CREP	
New Input.ind	FSPMM1	AREP	

Primitive Name	Source	Associated Parameters	Functions
Alarm Notification.ind	FSPMM1	AREP, Slot Number, Alarm Type, Seq Nr, Add Ack, Alarm Specifier, Alarm Data	
DP Master C11 Mode Changed.ind	FSPMM1	AREP, USIF State	
DP Master C11 Event.ind		Event, Add Info	
DP Master C11 Started.ind	FSPMM1	AREP	
DP Master C11 Stopped.ind	FSPMM1	AREP	
Abort.ind	FSPMM1	AREPI	
DP Master C11 Reject.ind	FSPMM1	AREP, Reason	
DP Master C11 Fault.ind	FSPMM1	(none)	
Load ARL DP Master C11.cnf(+)	FSPMM1	(none)	
Load ARL DP Master C11.cnf(-)	FSPMM1	Status	
Get ARL DP Master C11.cnf(+)	FSPMM1	Alarm Max, List of ARL Entries	
Get ARL DP Master C11.cnf(-)	FSPMM1	Status	
Load CRL DP Master C11.cnf(+)	FSPMM1	(none)	
Load CRL DP Master C11.cnf(-)	FSPMM1	Status	
Get CRL DP Master C11.cnf(+)	FSPMM1	List of CRL Entries	
Get CRL DP Master C11.cnf(-)	FSPMM1	Status	

6.5.2.1.2 Parameters of FSPMM1 Primitives

The parameters used with the primitives exchanged between the FSPMM1 and the AP-Context are described in FAL Type 3 Service Definition IEC 61158-5.

6.5.2.2 State Machine Description

This Machine is used to co-ordinate the Interactions between AP-Context and DMPMM1, MSCY1M, MSAC1M, MSAL1M and MMAC1. As the support of several State Machines for the communication with individual slave as well as timing constraints of the operation of slave require a consistent scheduling, this task is accomplished by FSPMM1 as well.

For each DP-Slave an individual set of state machines (MSCY1M, MSAL1M, MSAC1M) will be established which will be controlled by the Scheduler in the DP-Master (Class 1). The Scheduler will be controlled by the AP-Context with special services. The Slave-Handlers(MSCY1M) are triggered sequentially from the FSPMS-Scheduler to perform the necessary actions. The FSPMM1 notifies the operation modes Clear and Operate in fixed time intervals (Dx_Control_Timer) to the DP-Slaves. This occurs also at state transitions of FSPMM1.

The Scheduler provides four operation modes for the User:

- Offline
- Stop
- Clear
- Operate

The State Machine needs local variables which are described in clause a). Furthermore the machine uses timers and macros which are listed in clause b) and c).

a) Local Variables

Go_AClr

(Boolean)

State variable which indicates by the value True that after finishing the current Slave cycle the FSPMM1 has to force all DP-Slaves into a safe state (Scheduler-State = CLEAR).

Internal_Go_AClr

(Boolean)

Only after finishing the current Slave cycle the FSPMS has to go to state CLEAR if Go_AClr is True. During each cycle the state variable Internal_Go_AClr stores the actual value of Go_AClr. At the end of each cycle the value of Internal_Go_AClr is copied into Go_AClr.

Mark_Active

(Unsigned8)

Indicates the state of the local function Mark.

Range:

Possible values are:

0 => Mark is not active

1 => Mark is active, but not yet in execution

2 => Mark is active and in execution

Diag_Active

(Boolean)

Used to store the information if at least one of the DP-Slaves is not in the data exchange mode (True). The information is issued to the User with Mark.cnf.

UGC_Count

(Unsigned8)

Global_Control requests of the User are possible at any time during the FSPMM1 operation modes CLEAR and OPERATE. Confirmations of these requests will be passed to the User. The FSPMM1 uses the service Global_Control to send cyclic Global_Control requests with its actual operation mode to all assigned DP-Slaves. Confirmations of cyclic Global_Control requests have to be utilised by the FSPMM1. The FSPMM1 has to know which Global_Control.cnf belongs to which previously called Global_Control.req. It is not possible to resolve this relationship only by using DL services. Therefore the FSPMM1 must get this relationship information by using the sequential order of the received Global_Control confirmations. The counters UGC_Count and CGC_Count are used to store information about the order of the requests and also information about the order of the confirmations.

With the UGC_Count counter all User Global_Control requests are counted incrementally. The initial value is 0. It is decremented with each Global_Control.cnf. In addition the counter is related with the parameter Bus_Para.Max_User_Global_Control.

Type: Unsigned8

Range: 0 .. Bus_Para.Max_User_Global_Control

CGC_Count

(Unsigned8)

With this counter all cyclic Global_Control requests are counted which were started by the Scheduler. The initial value is 0. With each new cyclic Global_Control.req UGC_Count is incremented and its value is copied to CGC_Count. UGC_Count also is decremented with each new Global_Control.cnf. If this counter reaches the value 1, the actual Global_Control.cnf is a confirmation for a cyclic Global_Control.req

Range: 0 .. Bus_Para.Max_User_Global_Control

Bus_Accessible

(Boolean)

Indicates the actual state of the bus system. If services can be sent, its value is True.

Act_Rem_Add

(Unsigned8)

This variable stores the actually used station address of a DP-Slave.

Range: 0 .. 125

Act_msac1m_Max_L_sdu_len

(Unsigned8)

This variable is used to store the actual value of the parameter msac1m_Max_L_sdu_len.

Range: 0 .. 240

User_Sched_Reset

(Boolean)

This variable indicates whether the actual reset was started by the User (AP-Context) or by any other State Machine.

Act_New_Bus_Para

(Bus_Para)

This structure is used to save an actual bus parameter set passed from the AP-Context to FSPMM1 temporarily.

c) Timers**Min_SI_Interval_Timer**

This timer supervises the minimal access control time of the DP-Slaves. It is evaluated only within the FSPMM1 states CCLEAR and COPERATE.

Dx_Control_Interval_Timer

If this timer expires the broadcast service Global_Control is executed in order to inform the DP-Slave whether the DP-Master (Class 1) is in the state CLEAR or OPERATE. If the timer expires and the confirmation of the last Global_Control has not yet been received a physical bus access problem has occurred.

d) Macros**INDICATE_BUS_ACCESSIBILITY**

If Accessible = True Then Bus_Accessible=True

6.5.2.3 FSPMM1 State Table

The following definition for a state set is used in the state table below.

t_state = STOP or CLEAR or OPERATE

Table 380 - FSPMM1 State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	Load ARL DP Master Cl1.req (Alarm Max, List of ARL Entries) /valid parameters =>Load ARL DP Master Cl1.cnf(+)	POWER-ON
2	POWER-ON	Load ARL DP Master Cl1.req (Alarm Max, List of ARL Entries) /invalid parameters =>Status := NO Load ARL DP Master Cl1.cnf(-) (Status)	POWER-ON
3	POWER-ON	Get ARL DP Master Cl1.req /ARL loaded =>Get ARL DP Master Cl1.cnf(+) (Alarm Max, List of ARL Entries)	POWER-ON
4	POWER-ON	Get ARL DP Master Cl1.req /ARL not loaded =>Status := NO Get ARL DP Master Cl1.cnf(-) (Status)	POWER-ON
5	POWER-ON	Load CRL DP Master Cl1.req (List of CRL Entries) /valid parameters =>Load CRL DP Master Cl1.cnf(+)	POWER-ON
6	POWER-ON	Load CRL DP Master Cl1.req (List of CRL Entries) /invalid parameters =>Status := NO Load CRL DP Master Cl1.cnf(-) (Status)	POWER-ON
7	POWER-ON	Get CRL DP Master Cl1.req /CRL loaded =>Get CRL DP Master Cl1.cnf(+) (List of CRL Entries)	POWER-ON
8	POWER-ON	Get CRL DP Master Cl1.req /CRL not loaded =>Status := NO Get CRL DP Master Cl1.cnf(-) (Status)	POWER-ON
9	PON	FSPMM1_Init DP Master Cl1.req(Bus_Para) =>User_Init:=TRUE USER_ChangedCritical_Para := FALSE DMPMM1_Minit_DLL.req(Bus_Para)	INIT-DMPM
10	INIT-DMPM	DMPMM2_Minit_DLL.cnf /IsARExistent(MS1)=TRUE =>Act_Rem_Add := 0 MSAC1M_Minit_MSAC1.req(Rem_Add:=Act_Rem_Add)	INIT-AC1
11	INIT-DMPM	DMPMM2_Minit_DLL.cnf /IsARExistent(MS1)=FALSE =>Act_Rem_Add := 0 MSCY1M_Minit_MS0.req(Rem_Add:=Act_Rem_Add)	INIT-CY1
12	INIT-AC1	MSAC1M_Minit_MSAC1.cnf(Rem_Add) /Act_Rem_Add<125 =>Act_Rem_Add := Act_Rem_Add + 1 MSAC1M_Minit_MSAC1.req(Rem_Add:=Act_Rem_Add)	INIT-AC1
13	INIT-AC1	MSAC1M_Minit_MSAC1.cnf(Rem_Add) /Act_Rem_Add=125 =>Act_Rem_Add := 0 MSAL1M_Minit.req(Rem_Add:=Act_Rem_Add)	INIT-AL1
14	INIT-AL1	MSAL1M_Minit.cnf(Rem_Add) /Act_Rem_Add<125 =>Act_Rem_Add := Act_Rem_Add + 1 MSAL1M_Minit.req(Rem_Add:=Act_Rem_Add)	INIT-AL1
15	INIT-AL1	MSAL1M_Minit.cnf(Rem_Add) /Act_Rem_Add = 125 =>Act_Rem_Add := 0 MSCY1M_Minit_MS0.req(Rem_Add:=Act_Rem_Add)	INIT-CY1
16	INIT-CY1	MSCY1M_Minit_MS0.cnf(Rem_Add) /Act_Rem_Add<125 =>Act_Rem_Add := Act_Rem_Add + 1 MSCY1M_Minit_MS0.req(Rem_Add:=Act_Rem_Add)	INIT-CY1
17	INIT-CY1	MSCY1M_Minit_MS0.cnf(Rem_Add) /Act_Rem_Add=125 && IsARExistent(MM)=TRUE =>MMAC1_Minit_MM.req	INIT-MM1

#	Current State	Event /Condition =>Actions taken&&Output	Next State
18	INIT-CY1	MSCY1M_Minit_MS0.cnf(Rem_Add) /Act_Rem_Add=125 && IsARExistent(MM)=FALSE && USER_ChangedCritical_Para = FALSE =>Bus_accessible := FALSE if(USER_SetMode_Offline=FALSE) FSPMM1_Init DP Master C11.cnf else FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible) USER_SetMode_Offline:=FALSE endif	OFFLINE
19	INIT-MM1	MMAC1_Minit_MM.cnf /USER_ChangedCritical_Para = FALSE =>Bus_accessible := FALSE if(USER_SetMode_Offline=FALSE) FSPMM1_Init DP Master C11.cnf else FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible) USER_SetMode_Offline:=FALSE endif	OFFLINE
20	OFFLINE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State <> Stop && USIF_State <> Offline =>FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	OFFLINE
21	OFFLINE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Offline =>FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	OFFLINE
22	OFFLINE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Stop && Bus_Para invalid =>FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	OFFLINE
23	OFFLINE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Stop && Bus_Para valid =>Mark_Active := 0 Diag_Active := False DMPMM1_Set_Bus_Par.req(Bus_Para)	LDBP
24	OFFLINE	FSPMM1_Mark.req(AREP) =>FSPMM1_Mark.cnf(-)(AREP, Status:=NO)	OFFLINE
25	OFFLINE	FSPMM1_Global_Control.req(AREP_Sync_Command, Freeze_Command, Group_Select) =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	OFFLINE
26	OFFLINE	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) =>FSPMM1_Load_Bus_Par.cnf(-)(Status:=NO)	OFFLINE
27	OFFLINE	FSPMM1_Delete_SC DP Master C11.req(Address) =>DMPMM1_Delete_SC.req(Address)	DELSC-OFF
28	OFFLINE	FSPMM1_Read_Value DP Master C11.req(Variable) =>DMPMM1_Read_Value.req(Variable)	RDVAL-OFF
29	LDBP	DMPMM1_Set_Bus_Par.cnf /USER_ChangeCritical_Para = FALSE =>FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	STOP
30	DELSC-OFF	DMPMM1_Delete_SC.cnf =>FSPMM1_Delete_SC DP Master C11.cnf	OFFLINE
31	RDVAL-OFF	DMPMM1_Read_Value.cnf(Status, Value) =>FSPMM1_Read_Value DP Master C11.cnf(Status, Value)	OFFLINE
32	STOP	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Offline =>USER_SetMode_Offline:=TRUE	RESET
33	STOP	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Stop =>FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	STOP
34	STOP	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Operate =>FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	STOP
35	STOP	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Clear =>UGC_Count := 0 CGC_Count := 0 Bus_Accessible := True Go_AClr := Bus_Para.Error_Action_Flag Internal_Go_AClr := False Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) MSCY1M_Start_Slave_Handler.req(0 .. 125)	CLEAR

#	Current State	Event /Condition =>Actions taken&&Output	Next State
36	STOP	FSPMM1_Mark.req(AREP) =>FSPMM1_Mark.cnf(-)(AREP, Status:=NO)	STOP
37	STOP	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	STOP
38	STOP	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) /Bus_Para invalid =>FSPMM1_Load_Bus_Par.cnf(-)(Status:=IV)	STOP
39	STOP	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) /(Bus_Para valid) && (critical parameters unchanged) =>DMPMM1_Set_Bus_Par.req(Bus_Para)	LDBP-ST-CPU
40	STOP	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) /(Bus_Para valid) && (critical parameters changed) =>Act_New_Bus_Para := Bus_Para User_ChangedCritical_Para := TRUE	RESET
41	STOP	FSPMM1_Delete_SC DP Master C11.req(Address) =>DMPMM1_Delete_SC.req(Address)	DELSC-ST
42	STOP	FSPMM1_Read_Value DP Master C11.req(Variable) =>DMPMM1_Read_Value.req(Variable)	RDVAL-ST
43	LDBP-ST-CPU	DMPMM1_Set_Bus_Par.cnf =>FSPMM1_Load_Bus_Para.cnf(+)	STOP
44	DELSC-ST	DMPMM1_Delete_SC.cnf =>FSPMM1_Delete_SC DP Master C11.cnf	STOP
45	RDVAL-ST	DMPMM1_Read_Value.cnf(Status, Value) =>FSPMM1_Read_Value DP Master C11.cnf(Status, Value)	STOP
46	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Offline =>FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	CLEAR
47	CLEAR	FSPMM1_Mark DP Master C11.req(AREP) /Mark_Active = 0 =>Mark_Active := 1	CLEAR
48	CLEAR	FSPMM1_Mark DP Master C11.req(AREP) /Mark_Active <> 0 =>FSPMM1_Mark.cnf(-)(AREP, Status:=NO)	CLEAR
49	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Stop && Mark_Active = 0 =>Stop Dx_Control_Interval_Timer MSCY1M_Stop_Slave_Handler.req(0..125)	SLHSTP
50	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Stop && Mark_Active <> 0 =>Mark_Active := 0 Stop Dx_Control_Interval_Timer FSPMM1_Mark.cnf(-)(AREP; Status:=NO) MSCY1M_Stop_Slave_Handler.req(0..125)	SLHSTP
51	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Clear =>FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	CLEAR
52	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Operate && Go_AClr = False && Internal_Go_AClr = False && CGC_Count > 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2	SGC-CO-WGC
53	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Operate && Go_AClr = False && Internal_Go_AClr = False && CGC_Count = 0 =>UGC_Count := UGC_Count+1 CGC_Count := UGC_Count Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2 DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=0, Group_Select:=0)	SGC-CO
54	CLEAR	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Operate && (Go_AClr = True Internal_Go_AClr = True) =>FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	CLEAR
55	CLEAR	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /NOT received all MSCY1M_Start_Slave_Handler.cnf =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	CLEAR

#	Current State	Event /Condition =>Actions taken&&Output	Next State
56	CLEAR	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /received all MSCY1M_Start_Slave_Handler.cnf && CGC_Count = 0 && UGC_Count >= Bus_Para.Max_User_Global_Control =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	CLEAR
57	CLEAR	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /received all MSCY1M_Start_Slave_Handler.cnf && CGC_Count = 0 && UGC_Count < Bus_Para.Max_User_Global_Control =>UGC_Count := UGC_Count+1 DMPMM1_Global_Control.req(Rem_Add:=CREP.Rem_Add, Control_Command.1:=1, Group_Select)	CLEAR
58	CLEAR	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /received all MSCY1M_Start_Slave_Handler.cnf && CGC_Count > 0 && UGC_Count >= Bus_Para.Max_User_Global_Control+1 =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	CLEAR
59	CLEAR	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /received all MSCY1M_Start_Slave_Handler.cnf && CGC_Count > 0 && UGC_Count < Bus_Para.Max_User_Global_Control+1 =>UGC_Count := UGC_Count+1 DMPMM1_Global_Control.req(Rem_Add:=CREP.Rem_Add, Control_Command.1:=1, Group_Select)	CLEAR
60	CLEAR	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 0 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	CLEAR
61	CLEAR	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := 0	CLEAR
62	CLEAR	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count > 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := CGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	CLEAR
63	CLEAR	MSCY1M_Start_Slave_Handler.cnf(Rem_Add) /NOT received all MSCY1M_Start_Slave_Handler.cnf =>ignore	CLEAR
64	CLEAR	MSCY1M_Start_Slave_Handler.cnf(Rem_Add) /received all MSCY1M_Start_Slave_Handler.cnf =>Internal_Go_AClr := False Start_Min_SI_Interval_Timer(Bus_Para.Min_Slave_Interval) Set_Mode.cnf(Status:=OK, Bus_Accessible) MSCY1M_Cont_Slave_Handler.req(0..125, Output_Clear:=True)	CLEAR
65	CLEAR	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /NOT received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 2 =>Diag_Active := Diag_Active OR Diag Internal_Go_AClr := Internal_Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	CLEAR
66	CLEAR	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /NOT received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active <> 2 =>Internal_Go_AClr := Internal_Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	CLEAR

#	Current State	Event /Condition =>Actions taken&&Output	Next State
67	CLEAR	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 0 =>Internal_Go_AClr := Internal_Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	CCLEAR
68	CLEAR	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 1 =>Mark_Active:=2 Diag_Active := False Internal_Go_AClr := Internal_Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	CCLEAR
69	CLEAR	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 2 =>Diag_Active := Diag_Active OR Diag Internal_Go_AClr := Internal_Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr) Mark_Active:=0 FSPMM1_Mark.cnf(+)(AREP, Dia:=Diag_Active)	CCLEAR
70	CLEAR	Dx_Control_Interval_Timer expired /CGC_Count = 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) UGC_Count := UGC_Count+1 CGC_Count := UGC_Count DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=2, Group_Select:=0)	CLEAR
71	CLEAR	Dx_Control_Interval_Timer expired /CGC_Count > 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False	CLEAR
72	CLEAR	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) /Bus_Para invalid =>FSPMM1_Load_Bus_Par.cnf(-)(Status:=IV)	CLEAR
73	CLEAR	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) /(Bus_Para valid) && (critical parameters changed) =>FSPMM1_Load_Bus_Par.cnf(-)(Status:=NO)	CLEAR
74	CLEAR	FSPMM1_Load_Bus_Par DP Master C11.req(Bus_Para) /(Bus_Para valid) && (critical parameters unchanged) =>DMPMM1_Set_Bus_Par.req(Bus_Para)	LDBP-CL-CPU
75	CLEAR	FSPMM1_Delete_SC DP Master C11.req(Address) =>DMPMM1_Delete_SC.req(Address)	DELSC-CL
76	CLEAR	FSPMM1_Read_Value DP Master C11.req(Variable) =>DMPMM1_Read_Value.req(Variable)	RDVAL-CL
77	SLHSTP	MSCY1M_Stop_Slave_Handler.cnf(Rem_Add) /NOT received all MSCY1M_Stop_Slave_Handler.cnf =>ignore	SLHSTP
78	SLHSTP	MSCY1M_Stop_Slave_Handler.cnf(Rem_Add) /received all MSCY1M_Stop_Slave_Handler.cnf =>Set_Mode.cnf(Status:=OK, Bus_Accessible)	STOP
79	SGC-CO-WGC	Dx_Control_Interval_Timer expired =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	CLEAR
80	SGC-CO-WGC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count > 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := CGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	SGC-CO-WGC

#	Current State	Event /Condition =>Actions taken&&Output	Next State
81	SGC-CO-WGC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 =>INDICATE_BUS_ACCESSIBILITY CGC_Count := UGC_Count DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=0, Group_Select:=0)	SGC-CO
82	SGC-CO	Dx_Control_Interval_Timer expired =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False Set_Mode.cnf(Status:=OK, Bus_Accessible)	OPERATE
83	SGC-CO	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count > 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := CGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	SGC-CO
84	SGC-CO	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := 0 FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	OPERATE
85	CCLEAR	Min_SI_Interval_Timer expired =>Go_AClr := Internal_Go_AClr Internal_Go_AClr := False Start Min_SI_Interval_Timer(Bus_Para.Min_Slave_Interval) MSCY1M_Cont_Slave_Handler.req(0..125, Output_Clear:=True)	CLEAR
86	LDBP-CL-CPU	DMPMM1_Set_Bus_Par.cnf =>FSPMM1_Load_Bus_Para.cnf(+)	CLEAR
87	DELSC-CL	DMPMM1_Delete_SC.cnf =>FSPMM1_Delete_SC DP Master C11.cnf	CLEAR
88	RDVAL-CL	DMPMM1_Read_Value.cnf(Status, Value) =>FSPMM1_Read_Value DP Master C11.cnf(Status, Value)	CLEAR
89	OPERATE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State <> Clear && USIF_State <> Operate =>FSPMM1_Set_Mode DP Master C11.cnf(-)(AREP, Bus_accessible)	OPERATE
90	OPERATE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Operate =>FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	OPERATE
91	OPERATE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Clear && CGC_Count > 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2)	SGC-OC-WGC
92	OPERATE	FSPMM1_Set_Mode.req(USIF_State) /USIF_State = Clear && CGC_Count = 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) UGC_Count := UGC_Count+1 CGC_Count := UGC_Count DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=2, Group_Select:=0)	SGC-OC
93	OPERATE	FSPMM1_Mark DP Master C11.req(AREP) /Mark_Active = 0 =>Mark_Active := 1	OPERATE
94	OPERATE	FSPMM1_Mark DP Master C11.req(AREP) /Mark_Active <> 0 =>FSPMM1_Mark.cnf(-)(AREP, Status:=NO)	OPERATE
95	OPERATE	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /Control_Command.1 = 1 =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	OPERATE
96	OPERATE	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /Control_Command.1 = 0 && CGC_Count = 0 && UGC_Count >= Bus_Para.Max_User_Global_Control =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	OPERATE

#	Current State	Event /Condition =>Actions taken&&Output	Next State
97	OPERATE	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /Control_Command.1 = 0 && CGC_Count = 0 && UGC_Count < Bus_Para.Max_User_Global_Control =>UGC_Count := UGC_Count+1 DMPMM1_Global_Control.req(Rem_Add:=CREP.Rem_Add, Control_Command, Group_Select)	OPERATE
98	OPERATE	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /Control_Command.1 = 0 && CGC_Count > 0 && UGC_Count >= Bus_Para.Max_User_Global_Control+1 =>FSPMM1_Global_Control.cnf(-)(AREP, Status:=NO)	OPERATE
99	OPERATE	FSPMM1_Global_Control.req(AREP, Sync_Command, Freeze_Command, Group_Select) /Control_Command.1 = 0 && CGC_Count > 0 && UGC_Count < Bus_Para.Max_User_Global_Control+1 =>UGC_Count := UGC_Count+1 DMPMM1_Global_Control.req(Rem_Add:=CREP.Rem_Add, Control_Command, Group_Select)	OPERATE
100	OPERATE	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 0 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	OPERATE
101	OPERATE	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := 0	OPERATE
102	OPERATE	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count > 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := CGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	OPERATE
103	OPERATE	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /NOT received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 2 =>Diag_Active:=Diag_Active OR Diag Go_AClr := Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	OPERATE
104	OPERATE	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /NOT received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active <> 2 =>Go_AClr := Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	OPERATE
105	OPERATE	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 0 =>Go_AClr := Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	COPERATE
106	OPERATE	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 1 =>Mark_Active:=2 Diag_Active := False Go_AClr := Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr)	COPERATE

#	Current State	Event /Condition =>Actions taken&&Output	Next State
107	OPERATE	MSCY1M_Cont_Slave_Handler.cnf(Rem_Add, Diag, No_AClr) /received all MSCY1M_Cont_Slave_Handler.cnf && Mark_Active = 2 =>Diag_Active:=Diag_Active OR Diag Go_AClr := Go_AClr OR (Bus_Para.Error_Action_Flag AND NOT No_AClr) Mark_Active:=0 FSPMM1_Mark.cnf(+)(AREP, Dia:=Diag_Active)	COPERATE
108	OPERATE	Dx_Control_Interval_Timer expired /CGC_Count = 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) UGC_Count := UGC_Count+1 CGC_Count := UGC_Count DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=0, Group_Select:=0)	OPERATE
109	OPERATE	Dx_Control_Interval_Timer expired /CGC_Count > 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_accessible := False	OPERATE
110	OPERATE	FSPMM1_Load_Bus_Par DP Master Cl1.req(Bus_Para) /Bus_Para invalid =>FSPMM1_Load_Bus_Par.cnf(-)(Status:=IV)	OPERATE
111	OPERATE	FSPMM1_Load_Bus_Par DP Master Cl1.req(Bus_Para) /Bus_Para valid && (critical parameters changed) =>FSPMM1_Load_Bus_Par.cnf(-)(Status:=NO)	OPERATE
112	OPERATE	FSPMM1_Load_Bus_Par DP Master Cl1.req(Bus_Para) /Bus_Para valid && (critical parameters unchanged) =>DMPMM1_Set_Bus_Par.req(Bus_Para)	LDBP-OP- CPU
113	OPERATE	FSPMM1_Delete_SC DP Master Cl1.req(Address) =>DMPMM1_Delete_SC.req(Address)	DELSC-OP
114	OPERATE	FSPMM1_Read_Value DP Master Cl1.req(Variable) =>DMPMM1_Read_Value.req(Variable)	RDVAL-OP
115	SGC-OC- WGC	Dx_Control_Interval_Timer expired /Go_AClr = False =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False FSPMM1_Set_Mode DP Master Cl1.cnf(+)(AREP, Bus_accessible)	OPERATE
116	SGC-OC- WGC	Dx_Control_Interval_Timer expired /Go_AClr = True =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False	SGC-OC- WGC
117	SGC-OC- WGC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count > 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := CGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	SGC-OC- WGC
118	SGC-OC- WGC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 =>INDICATE_BUS_ACCESSIBILITY CGC_Count := UGC_Count DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=2, Group_Select:=0)	SGC-OC
119	SGC-OC	Dx_Control_Interval_Timer expired /Go_AClr = False =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False Set_Mode.cnf(Status:=OK, Bus_Accessible)	CLEAR
120	SGC-OC	Dx_Control_Interval_Timer expired /Go_AClr = True =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) Bus_Accessible := False FSPMM1_DP Master Cl1 Mode_Changed.ind(AREP, USIF_State:=Clear) MSCY1M_Cont_Slave_Handler.req(0..125, Output_Clear:=True)	CLEAR

#	Current State	Event /Condition =>Actions taken&&Output	Next State
121	SGC-OC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count > 1 =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := CGC_Count-1 FSPMM1_Global_Control.cnf(+)(AREP)	SGC-OC
122	SGC-OC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 && Go_AClr = False =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := 0 FSPMM1_Set_Mode DP Master C11.cnf(+)(AREP, Bus_accessible)	CLEAR
123	SGC-OC	DMPMM1_Global_Control.cnf(Rem_Add, Status) /CGC_Count = 1 && Go_AClr = True =>INDICATE_BUS_ACCESSIBILITY UGC_Count := UGC_Count-1 CGC_Count := 0 FSPMM1_DP Master C11 Mode_Changed.ind(AREP, USIF_State:=Clear) MSCY1M_Cont_Slave_Handler.req(0..125, Output_Clear:=True)	CLEAR
124	COPERATE	Min_SI_Interval_Timer expired /Go_AClr = False =>Start Min_SI_Interval_Timer (Bus_Para.Min_Slave_Interval) MSCY1M_Cont_Slave_Handler.req(0..125, Output_Clear:=False)	OPERATE
125	COPERATE	Min_SI_Interval_Timer expired /Go_AClr = True && CGC_Count > 0 =>Start Dx_Control_Interval_Timer (Bus_Para.Data_Control_Time/2) Start Min_SI_Interval_Timer (Bus_Para.Min_Slave_Interval)	SGC-OC- WGC
126	COPERATE	Min_SI_Interval_Timer expired /Go_AClr = True && CGC_Count = 0 =>Start Dx_Control_Interval_Timer(Bus_Para.Data_Control_Time/2) UGC_Count := UGC_Count+1 CGC_Count := UGC_Count Start Min_SI_Interval_Timer(Bus_Para.Min_Slave_Interval) DMPMM1_Global_Control.req(Rem_Add:=127, Control_Command:=2, Group_Select:=0)	SGC-OC
127	LDBP-OP- CPU	DMPMM1_Set_Bus_Par.cnf =>FSPMM1_Load_Bus_Para.cnf(+)	OPERATE
128	DELSC-OP	DMPMM1_Delete_SC.cnf =>FSPMM1_Delete_SC DP Master C11.cnf	OPERATE
129	RDVAL-OP	DMPMM1_Read_Value.cnf(Status, Value) =>FSPMM1_Read_Value DP Master C11.cnf(Status, Value)	OPERATE
130	ANY-STATE	DMPMM1_Fault.ind =>USER_sched_Reset := False FSPMM1_DP Master C11 Fault.ind	RESET
131	ANY-STATE	MSCY1M_Fault.ind =>USER_sched_Reset := False FSPMM1_DP Master C11 Fault.ind	RESET
132	ANY-STATE	MSAL1M_Fault.ind =>USER_sched_Reset := False FSPMM1_DP Master C11 Fault.ind	RESET
133	ANY-STATE	MSAC1M_Fault.ind =>USER_sched_Reset := False FSPMM1_DP Master C11 Fault.ind	RESET
134	ANY-STATE	MMAC1_Fault.ind =>USER_sched_Reset := False FSPMM1_DP Master C11 Fault.ind	RESET
135	ANY-STATE	FSPMM1_Reset DP Master C11.req =>USER_sched_Reset := True	RESET
136	RESET	/spontaneous =>Act_Rem_Add := 0 MSCY1M_Reset.req(Act_Rem_Add)	RESET-MS0
137	RESET-MS0	MSCY1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add<125 =>Act_Rem_Add := Act_Rem_Add+1 MSCY1M_Reset.req(Act_Rem_Add)	RESET-MS0

#	Current State	Event /Condition =>Actions taken&&Output	Next State
138	RESET-MS0	MSCY1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add=125 && IsARExistent(MS1)=TRUE =>Act_Rem_Add := 0 MSAC1M_Reset.req(Act_Rem_Add)	RESET-MS1
139	RESET-MS0	MSCY1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add=125 && IsARExistent(MS1)=FALSE && IsARExistent(MM1)=TRUE =>MMAC1_Reset.req	RESET-MM1
140	RESET-MS0	MSCY1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add=125 && IsARExistent(MS1)=FALSE && IsARExistent(MM1)=FALSE =>DMPMM1_Reset.req	RESET-DMPM
141	RESET-MS1	MSAC1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add=125 =>Act_Rem_Add := 0 MSAL1M_Reset.req(Act_Rem_Add)	RESET-MS1
142	RESET-MS1	MSAL1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add<125 =>Act_Rem_Add := Act_Rem_Add + 1 MSAL1M_Reset.req(Act_Rem_Add)	RESET-MS1
143	RESET-MS1	MSAL1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add = 125 && IsARExistent(MM1)=TRUE =>MMAC1_Reset.req	RESET-MM1
144	RESET-MS1	MSAL1M_Reset.cnf(Act_Rem_Add) /Act_Rem_Add = 125 && IsARExistent(MM1)=FALSE =>DMPMM1_Reset.req	RESET-DMPM
145	RESET-MM1	MMAC1_Reset.cnf =>DMPMM1_Reset.req	RESET-DMPM
146	RESET-DMPM	DMPMM1_Reset.cnf /USER_SetMode_Offline = TRUE USER_ChangedCritical_Para = TRUE	INIT-DMPM
147	RESET-DMPM	DMPMM1_Reset.cnf /USER_SetMode_Offline = FALSE && USER_ChangedCritical_Para = FALSE =>if(USER_sched_Reset = TRUE) FSPMM1_Reset DP Master C1.cnf endif	PON
148	INIT-CY1	MSCY1M_Init_MS0.cnf(Rem_Add) /Act_Rem_Add=125 && IsARExistent(MM)=FALSE && USER_ChangedCritical_Para = TRUE =>DMPMM1_Set_Bus_Par.req(Bus_Para:=Act_New_Bus_Para)	LDBP
149	INIT-MM1	MMAC1_Init_MM.cnf /USER_ChangedCritical_Para = TRUE =>DMPMM1_Set_Bus_Par.req(Bus_Para:=Act_New_Bus_Para)	LDBP
150	LDBP	DMPMM1_Set_Bus_Par.cnf /USER_ChangeCritical_Para = TRUE =>FSPMM1_Load_Bus_Para.cnf(+)	STOP
151	t state	FSPMM1_Abort DP Master C1.req(AREP) /AREP.AR_Type=MS0 =>MSCY1M_Abort.req(Rem_Add:=CREP.Rem_Add)	SAME
152	t state	FSPMM1_Abort DP Master C1.req(AREP) /AREP.AR_Type=MS1 =>MSAC1M_Abort.req(Rem_Add:=CREP.Rem_Add)	SAME
153	t state	FSPMM1_Abort DP Master C1.req(AREP) /AREP.AR_Type=MM1 =>MMAC1_Abort.req(Rem_Add:=CREP.Rem_Add)	SAME
154	t state	FSPMM1_Get_Slave_Diag.req(AREP, CREP) /AREP.AR_Type=MS0 =>MSCY1M_Get_Slave_Diag.req(CREP.Rem_Add)	SAME
155	t state	MSCY1M_Get_Slave_Diag.cnf(+)(Rem_Add, Diag_Data) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Get_Slave_Diag.cnf(+)(AREP, CREP, Diag_Data)	SAME
156	t state	MSCY1M_Get_Slave_Diag.cnf(-)(Rem_Add) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Get_Slave_Diag.cnf(-)(AREP, CREP)	SAME
157	t state	FSPMM1_Set_Output.req(AREP, CREP, Slot_Number, Output_Data) /AREP.AR_Type=MS0 =>MSCY1M_Set_Output.req(Rem_Add:=CREP.Rem_Add, Slot_Number, Output_Data)	SAME
158	t state	MSCY1M_Set_Output.cnf(+)(Rem_Add, Slot_Number) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Set_Output.cnf(+)(AREP, CREP, Slot_Number)	SAME

#	Current State	Event /Condition =>Actions taken&&Output	Next State
159	t state	MSCY1M_Set_Output.cnf(-)(Rem_Add, Slot_Number) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Set_Output.cnf(-)(AREP, CREP, Slot_Number)	SAME
160	t state	FSPMM1_Get_Input.req(AREP, CREP, Slot_Number) /AREP.AR_Type=MS0 =>MSCY1M_Get_Input.req(Rem_Add:=CREP.Rem_Add, Slot_Number)	SAME
161	t state	MSCY1M_Get_Input.cnf(+)(Rem_Add, Slot_Number, Input_Data) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Get_Input.cnf(+)(AREP, CREP, Slot_Number, Input_Data)	SAME
162	t state	MSCY1M_Get_Input.cnf(-)(Rem_Add, Slot_Number) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Get_Input.cnf(-)(AREP, CREP, Slot_Number)	SAME
163	t state	FSPMM1_Read.req(AREP, Slot_Number, Index, Length) /AREP.AR_Type=MS1 =>MSAC1M_Read.req(Rem_Add:=CREP.Rem_Add, Slot_Number, Index, Length)	SAME
164	t state	MSAC1M_Read.cnf(+)(Rem_Add, Length, Data) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Read.cnf(+)(AREP, Length, Data)	SAME
165	t state	MSAC1M_Read.cnf(-)(Rem_Add, Error_Decode, Error_Code_1, Error_Code_2) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Read.cnf(-)(AREP, Rem_Add, Error_Decode, Error_Code_1, Error_Code_2)	SAME
166	t state	FSPMM1_Write.req(AREP, Slot_Number, Index, Length, Data) /AREP.AR_Type=MS1 =>MSAC1M_Write.req(Rem_Add:=CREP.Rem_Add, Slot_Number, Index, Length, Data)	SAME
167	t state	MSAC1M_Write.cnf(+)(Rem_Add, Length) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Write.cnf(+)(AREP, Length)	SAME
168	t state	MSAC1M_Write.cnf(-)(Rem_Add, Error_Decode, Error_Code_1, Error_Code_2) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Write.cnf(-)(AREP, Rem_Add, Error_Decode, Error_Code_1, Error_Code_2)	SAME
169	t state	FSPMM1_Alarm_Ack.req(AREP, Slot_Number, Alarm_Type, Seq_Nr) /AREP.AR_Type=MS1 =>MSAC1M_Alarm_Ack.req(Rem_Add:=CREP.Rem_Add, Slot_Number, Alarm_Type, Seq_Nr)	SAME
170	t state	MSAC1M_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Alarm_Ack.cnf(+)(AREP, Slot_Number, Alarm_Type, Seq_Nr)	SAME
171	t state	MSAC1M_Alarm_Ack.cnf(-)(Rem_Add) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Alarm_Ack.cnf(-)(AREP, Slot Number, Alarm Type, Seq Nr)	SAME
172	t state	FSPMM1_Get_Master_Diag.rsp(+)(AREP, Diagnosis_Data) /AREP.AR_Type=MM1 =>MMAC1_Get_Master_Diag.rsp(Status:=OK, Diagnosis_Data)	SAME
173	t state	FSPMM1_Get_Master_Diag.rsp(-)(AREP, Status) /AREP.AR_Type=MM1 =>MMAC1_Get_Master_Diag.rsp(Status)	SAME
174	t state	MMAC1_Get_Master_Diag.ind(Identifier) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Get_Master_Diag.ind(AREP, Mdiag_Identifier:=Identifier)	SAME
175	t state	FSPMM1_Start_Seq.rsp(+)(AREP, Max_Len_Data_Unit) /AREP.AR_Type=MM1 =>MMAC1_Start_Seq.rsp(Status:=OK, Max_Length_Data_Unit)	SAME
176	t state	FSPMM1_Start_Seq.rsp(-)(AREP, Status) /AREP.AR_Type=MM1 =>MMAC1_Start_Seq.rsp(Status)	SAME
177	t state	MMAC1_Start_Seq.ind(Req_Add, Area_Code, Timeout) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Start_Seq.ind(AREP, Area_Code, Timeout)	SAME
178	t state	FSPMM1_Download.rsp(+)(AREP) /AREP.AR_Type=MM1 =>MMAC1_Download.rsp(Status:=OK)	SAME
179	t state	FSPMM1_Download.rsp(-)(AREP, Status) /AREP.AR_Type=MM1 =>MMAC1_Download.rsp(Status)	SAME
180	t state	MMAC1_Download.ind(Req_Add, Area_Code, Address_Offset, Data) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Download.ind(AREP, Area_Code, Add_Offset, Data)	SAME

#	Current State	Event /Condition =>Actions taken&&Output	Next State
181	t state	FSPMM1_Upload.rsp(+)(AREP, Data) /AREP.AR_Type=MM1 =>MMAC1_Upload.rsp(Status:=OK, Data)	SAME
182	t state	FSPMM1_Upload.rsp(-)(AREP, Status) /AREP.AR_Type=MM1 =>MMAC1_Upload.rsp(Status)	SAME
183	t state	MMAC1_Upload.ind(Req_Add, Area_Code, Address_Offset, Data_Length) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Upload.ind(AREP, Area_Code, Add_Offset, Data_Len:=Data_Length)	SAME
184	t state	FSPMM1_End_Seq.rsp(+)(AREP) /AREP.AR_Type=MM1 =>MMAC1_End_Seq.rsp(Status:=OK)	SAME
185	t state	FSPMM1_End_Seq.rsp(-)(AREP, Status) /AREP.AR_Type=MM1 =>MMAC1_End_Seq.rsp(Status)	SAME
186	t state	MMAC1_End_Seq.ind(Req_Add) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_End_Seq.ind(AREP)	SAME
187	t state	FSPMM1_Act_Param.rsp(+)(AREP) /AREP.AR_Type=MM1 =>MMAC1_Act_Param.rsp(Status:=OK)	SAME
188	t state	FSPMM1_Act_Param.rsp(-)(AREP, Status) /AREP.AR_Type=MM1 =>MMAC1_Act_Param.rsp(Status)	SAME
189	t state	MMAC1_Act_Param.ind(Area_Code, Activate) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Act_Param.ind(AREP, Area_Code, Activate)	SAME
190	t state	MMAC1_Act_Para_Brct.ind(Area_Code) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Act_Para_Brct.ind(AREP, Area_Code)	SAME
191	ANY-STATE	DMPMM1_Event.ind(Event, Add_Info) =>FSPMM1_DP_Master_C11_Event.ind(Event, Add_Info)	SAME
192	t state	MSAC1M_Reject.ind(Rem_Add, Status) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_DP_Master_C11_Reject.ind(AREP, Reason:=Status)	SAME
193	t state	MSAL1M_Started.ind(Rem_Add, Alarm_Limit) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_DP_Master_C11_Started.ind(AREP, Alarm_Limit)	SAME
194	t state	MSAL1M_Stopped.ind(Rem_Add) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_DP_Master_C11_Stopped.ind(AREP)	SAME
195	t state	MSAL1M_Alarm_Notification.ind(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr, Alarm_Specifier, Add_Ack, Alarm_Data) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_Alarm_Notification.ind(AREP, Alarm_Type, Slot_Number, Seq_Nr, Alarm_Specifier, Add_Ack, Alarm_Data)	SAME
196	t state	MSCY1M_New_Slave_Diag.ind(Rem_Add) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_New_Slave_Diag.ind(AREP, CREP)	SAME
197	t state	MSCY1M_New_Input.ind(Rem_Add) /SetContext(Rem_Add, Service)=TRUE =>FSPMM1_New_Input.ind(AREP, CREP)	SAME

6.5.2.4 Functions

Function Name	Description
SetContext (Rem_Add, Service-Id)	This function checks if there exist an entry for the inputs Rem_Add and Service in the ARL and related CRL of the DP-Master (Class 1). A) If there exist an entry than it returns TRUE and sets the local context according to the current CREP and AREP. B) Otherwise it returns FALSE.
IsARExistent (AR)	This function checks if there exist an entry with ARtype=AR.

6.5.3 FSPMM2

6.5.3.1 Primitive Definitions

6.5.3.1.1 Primitives Exchanged between AP-Context and FSPMM2

Table 381 - Primitives issued by AP-Context to FSPMM2

Primitive Name	Source	Associated Parameters	Functions
Init DP Master Cl2.req	AP-Context	Bus Para	Refer to FAL Type 3 Service Definition IEC 61158-5 and IEC 61158-3
Reset DP Master Cl2.req	AP-Context	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Abort.req	AP-Context	AREP Subnet, Instance Reason Code	
Read Slave Diag.req	AP-Context	AREP	
Read Output.req	AP-Context	AREP	
Read Input.req	AP-Context	AREP	
Get Cfg.req	AP-Context	AREP	
Set Slave Add.req	AP-Context	AREP, New Slave Add, Ident Number, No Add Chg, Rem Slave Data	
Initiate.req	AP-Context	AREP, Send Timeout, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
Read.req	AP-Context	AREP, Slot Number, Index, Length	
Write.req	AP-Context	AREP, Slot Number, Index, Length, Data	
Data Transport.req	AP-Context	AREP, Slot Number, Index, Length, Data	
Get Master Diag.req	AP-Context	AREP, MDiag Identifier	
Start Seq.req	AP-Context	AREP, Area Code, Timeout	
Download.req	AP-Context	AREP, Area Code, Add Offset, Data	

Primitive Name	Source	Associated Parameters	Functions
Upload.req	AP-Context	AREP, Area Code, Add Offset, Data Len	
End Seq.req	AP-Context	AREP	
Act Param.req	AP-Context	AREP, Area Code, Activate	
Act Para Brct.req	AP-Context	AREP, Area Code	
Load ARL DP Master Cl2.req	AP-Context	List of ARL Entries	
Get ARL DP Master Cl2.req	AP-Context	(none)	
Load CRL DP Master Cl2.req	AP-Context	List of CRL Entries	
Get CRL DP Master Cl2.req	AP-Context	(none)	

Table 382 - Primitives issued by FSPMM2 to AP-Context

Primitive Name	Source	Associated Parameters	Functions
Init DP Master Cl2.cnf	FSPMM2	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	FSPMM2	(none)	
Read Slave Diag.cnf(+)	FSPMM2	AREP, Diag Data	
Read Slave Diag.cnf(-)	FSPMM2	AREP, Status	
Get Cfg.cnf(+)	FSPMM2	AREP, Cfg Data	
Get Cfg.cnf(-)	FSPMM2	AREP, Status	
Read Output.cnf(+)	FSPMM2	AREP, Output Data	
Read Output.cnf(-)	FSPMM2	AREP, Status	
Read Input.cnf(+)	FSPMM2	AREP, Input Data	
Read Input.cnf(-)	FSPMM2	AREP, Status	
Set Slave Add.cnf(+)	FSPMM2	AREP	
Set Slave Add.cnf(-)	FSPMM2	AREP, Status	
Initiate.cnf(+)	FSPMM2	AREP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
Initiate.cnf(-)	FSPMM2	AREP, Error Decode, Error Code 1 Error Code 2	
Read.cnf(+)	FSPMM2	AREP, Length, Data	
Read.cnf(-)	FSPMM2	AREP, Error Decode, Error Code 1 Error Code 2	
Write.cnf(+)	FSPMM2	AREP, Length	
Write.cnf(-)	FSPMM2	AREP, Error Decode, Error Code 1 Error Code 2	

Primitive Name	Source	Associated Parameters	Functions
Data Transport.cnf(+)	FSPMM2	AREP, Length, Data	
Data Transport.cnf(-)	FSPMM2	AREP, Error Decode, Error Code 1 Error Code 2	
Get Master Diag.cnf(+)	FSPMM2	AREP, Diagnosis Data	
Get Master Diag.cnf(-)	FSPMM2	AREP, Status	
Start Seq.cnf(+)	FSPMM2	AREP, Max Len Data Unit	
Start Seq.cnf(-)	FSPMM2	AREP, Status	
Download.cnf(+)	FSPMM2	AREP	
Download.cnf(-)	FSPMM2	AREP, Status	
Upload.cnf(+)	FSPMM2	AREP, Data	
Upload.cnf(-)	FSPMM2	AREP, Status	
End Seq.cnf(+)	FSPMM2	AREP	
End Seq.cnf(-)	FSPMM2	AREP, Status	
Act Param.cnf(+)	FSPMM2	AREP	
Act Param.cnf(-)	FSPMM2	AREP, Status	
Act Para Brct.cnf(+)	FSPMM2	AREP	
Act Para Brct.cnf(-)	FSPMM2	AREP, Status	
Event.ind	FSPMM2	Event, Add Info	
DP Master Cl2 Reject.ind	FSPMM2	AREP, Reason	
Abort.ind	FSPMM2	AREP, Locally Generated, Subnet, Instance, Reason_Code, Additional_Detail	
DP Master Cl2 Fault.ind	FSPMM2	(none)	
Load ARL DP Master Cl2.cnf(+)	FSPMM2	(none)	
Load ARL DP Master Cl2.cnf(-)	FSPMM2	Status	
Get ARL DP Master Cl2.cnf(+)	FSPMM2	List of ARL Entries	
Get ARL DP Master Cl2.cnf(-)	FSPMM2	Status	
Load CRL DP Master Cl2.cnf(+)	FSPMM2	(none)	
Load CRL DP Master Cl2.cnf(-)	FSPMM2	Status	
Get CRL DP Master Cl2.cnf(+)	FSPMM2	List of CRL Entries	
Get CRL DP Master Cl2.cnf(-)	FSPMM2	Status	

6.5.3.1.2 Parameters of FSPMM2 Primitives

The parameters used with the primitives exchanged between the FSPMM2 and the AP-Context are described in FAL Type 3 Service Definition IEC 61158-5.

6.5.3.2 State Machine Description

This Machine is used to co-ordinate the Interactions between AP-Context and DMPMM1, MSCY1M, MSAC1M, MSAL1M and MMAC1. As the support of several State Machines for the communication with individual slave as well as timing constrains of the operation of slave require a consistent scheduling, this task is accomplished by FSPMM1 as well.

6.5.3.3 FSPMM2 State Table

Table 383 - FSPMM2 State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	Load ARL DP Master Ci2.req (List of ARL Entries) /valid parameters =>Load ARL DP Master Ci2.cnf(+)	POWER-ON
2	POWER-ON	Load ARL DP Master Ci2.req (List of ARL Entries) /invalid parameters =>Status := NO Load ARL DP Master Ci2.cnf(-) (Status)	POWER-ON
3	POWER-ON	Get ARL DP Master Ci2.req /ARL loaded =>Get ARL DP Master Ci2.cnf(+) (List of ARL Entries)	POWER-ON
4	POWER-ON	Get ARL DP Master Ci2.req /ARL not loaded =>Status := NO Get ARL DP Master Ci2.cnf(-) (Status)	POWER-ON
5	POWER-ON	Load CRL DP Master Ci2.req (List of CRL Entries) /valid parameters =>Load CRL DP Master Ci2.cnf(+)	POWER-ON
6	POWER-ON	Load CRL DP Master Ci2.req (List of CRL Entries) /invalid parameters =>Status := NO Load CRL DP Master Ci2.cnf(-) (Status)	POWER-ON
7	POWER-ON	Get CRL DP Master Ci2.req /CRL loaded =>Get CRL DP Master Ci2.cnf(+) (List of CRL Entries)	POWER-ON
8	POWER-ON	Get CRL DP Master Ci2.req /CRL not loaded =>Status := NO Get CRL DP Master Ci2.cnf(-) (Status)	POWER-ON
9	PON	FSPMM2_Init DP Master Ci2 .req(Bus_Para) => StoreNumberIssuedMinit() DMPMM2_Minit_FDL.req(Bus_Para) MMAC_Minit_MM.req MSAC2M_Minit_MS2.req(First_MS2_AREP ..Last_MS2_AREP)	INIT-DLM
10	INIT-DLM	DMPMM2_Minit_FDL.cnf =>ignore	INIT-DLM
11	INIT-DLM	MMAC2_Minit_MM.cnf =>ignore	INIT-DLM
12	INIT-DLM	MSAC2M_Minit_MS2.cnf (CREP) /IsLastWaitingMinit()=FALSE =>ignore	INIT-DLM
13	INIT-DLM	MSAC2M_Minit_MS2.cnf (CREP) /IsLastWaitingMinit()=TRUE =>FSPMM2_Init DP Master Ci2 .cnf	RUN
14	RUN	Any FSPMM2.req /wrong AREP.AR_Type =>ignore	RUN
15	RUN	Any MMAC1.cnf /SetContext(Rem_add, Service) = FALSE =>ignore	RUN
16	RUN	Any MSAC2M.req /SetContext(CREP, Service) = FALSE =>ignore	RUN
17	RUN	Any DMPMM2.req /SetContext(Rem_add, Service) = FALSE =>ignore	RUN
18	RUN	FSPMM2_Abort.req(AREP, Subnet, Instance, Reason_Code) /AREP.AR_Type = MS0 =>DMPMM2_Abort.req	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
19	RUN	FSPMM2_Abort.req(AREP, Subnet, Instance, Reason_Code) /AREP.AR_Type = MM1 =>MMAC2_Abort.req	RUN
20	RUN	FSPMM2_Abort.req(AREP, Subnet, Instance, Reason_Code) /AREP.AR_Type = MS2 =>MSAC2M_Abort.req(CREP, Subnet, Instance, Reason_Code)	RUN
21	RUN	MSAC2M_Abort.ind(CREP, Locally_Generated, Subnet, Instance, Reason_Code, Abort_Detail) /SetContext(CREP, Service) = TRUE =>FSPMM2_Abort.ind(AREP, Locally Generated, Subnet, Instance, Reason Code, Additional Detail)	RUN
22	RUN	DMPMM2_Event.ind(Event, Add_info) =>FSPMM2_Event.ind(Event=Event/Fault, Add_Info=Add_info)	RUN
23	RUN	FSPMM2_Read_Slave_Diag.req(AREP) /AREP.AR_Type = MS0 =>DMPMM2_Read_Slave_Diag.req(CREP.Rem_Add)	RUN
24	RUN	DMPMM2_Read_Slave_Diag.cnf(+)(Rem_Add, Diag_Data) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Read_Slave_Diag.cnf(+)(AREP, Diag_Data)	RUN
25	RUN	DMPMM2_Read_Slave_Diag.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Read_Slave_Diag.cnf(-)(AREP, Status=Status)	RUN
26	RUN	FSPMM2_Get_Cfg.req(AREP) /AREP.AR_Type = MS0 =>DMPMM2_Get_Cfg.req(CREP.Rem_Add)	RUN
27	RUN	DMPMM2_Get_Cfg.cnf(+)(Rem_Add, Cfg_Data) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Get_Cfg.cnf(+)(AREP, Cfg_Data)	RUN
28	RUN	DMPMM2_Get_Cfg.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Get_Cfg.cnf(-)(AREP, Status)	RUN
29	RUN	FSPMM2_Read_Input.req(AREP) /AREP.AR_Type = MS0 =>DMPMM2_Read_Input.req(CREP.Rem_Add)	RUN
30	RUN	DMPMM2_Read_Input.cnf(+)(Rem_Add, Inp_Data) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Read_Input.cnf(+)(AREP, Inp_Data)	RUN
31	RUN	DMPMM2_Read_Input.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Read_Input.cnf(-)(AREP, Status)	RUN
32	RUN	FSPMM2_Read_Output.req(AREP) /AREP.AR_Type = MS0 =>DMPMM2_Read_Output.req(CREP.Rem_Add)	RUN
33	RUN	DMPMM2_Read_Output.cnf(+)(Rem_Add, Outp_Data) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Read_Output.cnf(+)(AREP, Outp_Data)	RUN
34	RUN	DMPMM2_Read_Output.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Read_Output.cnf(-)(AREP, Status)	RUN
35	RUN	FSPMM2_Set_Slave_Add.req(AREP, New_Slave_Add, Ident_Number, No_Add_Chg, Rem_Slave_Data) /AREP.AR_Type = MS0 =>DMPMM2_Set_Slave_Add.req(CREP.Rem_Add, New_Slave_Add, Ident_Number, No_Add_Chg, Rem_Slave_Data)	RUN
36	RUN	DMPMM2_Set_Slave_Add.cnf(+)(Rem_Add) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Set_Slave_Add.cnf(+)(AREP)	RUN
37	RUN	DMPMM2_Set_Slave_Add.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Set_Slave_Add.cnf(-)(AREP, Status=Status)	RUN
38	RUN	Any DMPMM2.req /SetContext(Rem_add, Service) = FALSE =>ignore	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
39	RUN	FSPMM2_Initiate.req(AREP, Send Timeout, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param) /AREP.AR_Type = MS2 =>MSAC2M_Initiate.req(CREP, Send Timeout, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param)	RUN
40	RUN	MSAC2M_Initiate.cnf(+)(CREP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param) /SetContext(CREP, Service) = TRUE =>FSPMM2_Initiate.cnf(+)(AREP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param)	RUN
41	RUN	MSAC2M_Initiate.cnf(-)(CREP, Error Decode, Error Code 1, Error Code 2) /SetContext(CREP, Service) = TRUE =>FSPMM2_Initiate.cnf(-)(AREP, Error Decode, Error Code 1, Error Code 2)	RUN
42	RUN	FSPMM2_Read.req(AREP, Slot Number, Index, Length) /AREP.AR_Type = MS2 =>MSAC2M_Read.req(CREP, Slot Number, Index, Length)	RUN
43	RUN	MSAC2M_Read.cnf(+)(CREP, Length, Data) /SetContext(CREP, Service) = TRUE =>FSPMM2_Read.cnf(+)(AREP, Length, Data)	RUN
44	RUN	MSAC2M_Read.cnf(-)(CREP, Error Decode, Error Code 1, Error Code 2) /SetContext(CREP, Service) = TRUE =>FSPMM2_Read.cnf(-)(AREP, Error Decode, Error Code 1, Error Code 2)	RUN
45	RUN	FSPMM2_Write.req(AREP, Slot Number, Index, Length, Data) /AREP.AR_Type = MS2 =>MSAC2M_Write.req(CREP, Slot Number, Index, Length, Data)	RUN
46	RUN	MSAC2M_Write.cnf(+)(CREP, Length) /SetContext(CREP, Service) = TRUE =>FSPMM2_Write.cnf(+)(AREP, Length)	RUN
47	RUN	MSAC2M_Write.cnf(-)(CREP, Error Decode, Error Code 1, Error Code 2) /SetContext(CREP, Service) = TRUE =>FSPMM2_Write.cnf(-)(AREP, Error Decode, Error Code 1, Error Code 2)	RUN
48	RUN	FSPMM2_Data_Transport.req(AREP, Slot Number, Index, Length, Data) /AREP.AR_Type = MS2 =>MSAC2M_Data_Transport.req(CREP, Slot Number, Index, Length, Data)	RUN
49	RUN	MSAC2M_Data_Transport.cnf(+)(CREP, Length, Data) /SetContext(CREP, Service) = TRUE =>FSPMM2_Data_Transport.cnf(+)(AREP, Length, Data)	RUN
50	RUN	MSAC2M_Data_Transport.cnf(-)(CREP, Error Decode, Error Code 1, Error Code 2) /SetContext(CREP, Service) = TRUE =>FSPMM2_Data_Transport.cnf(-)(AREP, Error Decode, Error Code 1, Error Code 2)	RUN
51	RUN	MSAC2M_Closed.ind(CREP) /SetContext(CREP, Service) = TRUE =>FSPMM2_DP_Master_Ci2_Closed.ind(AREP)	RUN
52	RUN	MMAC2_Reject.ind(Rem_Add, Reason_Code) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_DP_Master_Ci2_Reject.ind(AREP, Reason)	RUN
53	RUN	FSPMM2_Get_Master_Diag.req(AREP, MDiag Identifier) /AREP.AR_Type = MM1 =>MMAC2_Get_Master_Diag.req(AREP, MDiag Identifier)	RUN
54	RUN	MMAC2_Get_Master_Diag.cnf(+)(Rem_Add, Diagnosis Data) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Get_Master_Diag.cnf(+)(AREP, Diagnosis Data)	RUN
55	RUN	MMAC2_Get_Master_Diag.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Get_Master_Diag.cnf(-)(AREP, Status)	RUN
56	RUN	FSPMM2_Start_Seq.req(AREP, Area Code, Timeout) /AREP.AR_Type = MM1 =>MMAC2_Start_Seq.req(AREP, Area Code, Timeout)	RUN
57	RUN	MMAC2_Start_Seq.cnf(+)(Rem_Add, Max Len Data Unit) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Start_Seq.cnf(+)(AREP, Max Len Data Unit)	RUN
58	RUN	MMAC2_Start_Seq.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Start_Seq.cnf(-)(AREP, Status)	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
59	RUN	FSPMM2_End_Seq.req(AREP) /AREP.AR_Type = MM1 =>MMAC2_End_Seq.req(AREP)	RUN
60	RUN	MMAC2_End_Seq.cnf(+)(Rem_Add) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_End_Seq.cnf(+)(AREP)	RUN
61	RUN	MMAC2_End_Seq.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_End_Seq.cnf(-)(AREP, Status)	RUN
62	RUN	FSPMM2_Download.req(AREP, Area Code, Add Offset, Data) /AREP.AR_Type = MM1 =>MMAC2_Download.req(AREP, Area Code, Add Offset, Data)	RUN
63	RUN	MMAC2_Download.cnf(+)(Rem_Add) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Download.cnf(+)(AREP)	RUN
64	RUN	MMAC2_Download.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Download.cnf(-)(AREP, Status)	RUN
65	RUN	FSPMM2_Upload.req(AREP, Area Code, Add Offset, Data Len) /AREP.AR_Type = MM1 =>MMAC2_Upload.req(AREP, Area Code, Add Offset, Data Len)	RUN
66	RUN	MMAC2_Upload.cnf(+)(Rem_Add, Data) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Upload.cnf(+)(AREP, Data)	RUN
67	RUN	MMAC2_Upload.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Upload.cnf(-)(AREP, Status)	RUN
68	RUN	FSPMM2_Act_Param.req(AREP, Area Code, Activate) /AREP.AR_Type = MM1 =>MMAC2_Act_Param.req(AREP, Area Code, Activate)	RUN
69	RUN	MMAC2_Act_Param.cnf(+)(Rem_Add) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Act_Param.cnf(+)(AREP)	RUN
70	RUN	MMAC2_Act_Param.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Act_Param.cnf(-)(AREP, Status)	RUN
71	RUN	FSPMM2_Act_Param_Brct.req(AREP, Area Code) /AREP.AR_Type = MM2 =>MMAC2_Act_Param_Brct.req(AREP, Area Code, Activate)	RUN
72	RUN	MMAC2_Act_Param_Brct.cnf(+)(Rem_Add) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Act_Param_Brct.cnf(+)(AREP)	RUN
73	RUN	MMAC2_Act_Param_Brct.cnf(-)(Rem_Add, Status) /SetContext(Rem_add, Service) = TRUE =>FSPMM2_Act_Param_Brct.cnf(-)(AREP, Status)	RUN
74	any state	FSPMM2_Reset_DP_Master_Ci2.req =>StoreNumberIssuedResets() FSPMM2_UserReset = TRUE DMPMM2_Reset.req MMAC2_Reset.req MSAC2M_Reset.req(First_MS2_AREP ..Last_MS2_AREP)	WAIT- RESET
75	any state	DMPMM2_Fault.ind =>StoreNumberIssuedResets() FSPMM2_UserReset = FALSE DMPMM2_Reset.req MMAC2_Reset.req MSAC2M_Reset.req(First_MS2_AREP ..Last_MS2_AREP) FSPMM2_DP_Master_Ci2_Fault.ind	WAIT- RESET
76	any state	MMAC2_Fault.ind =>StoreNumberIssuedResets() FSPMM2_UserReset = FALSE DMPMM2_Reset.req MMAC2_Reset.req MSAC2M_Reset.req(First_MS2_AREP ..Last_MS2_AREP) FSPMM2_DP_Master_Ci2_Fault.ind	WAIT- RESET
77	WAIT-RESET	DMPMM2_Reset.cnf =>no action	WAIT- RESET

#	Current State	Event /Condition =>Actions taken&&Output	Next State
78	WAIT-RESET	MMAC2_Reset.cnf =>no action	WAIT-RESET
79	WAIT-RESET	MSAC2M_Reset.cnf (CREP) /IsLastWaitingReset()=FALSE =>no action	WAIT-RESET
80	WAIT-RESET	MSAC2M_Reset.cnf (CREP) /IsLastWaitingReset()=TRUE AND FSPMM2_UserReset = TRUE =>FSPMM2_Reset DP Master C12 .cnf	PON
81	WAIT-RESET	MSAC2M_Reset.cnf (CREP) /IsLastWaitingReset()=TRUE AND FSPMM2_UserReset = FALSE =>no action	PON

6.5.3.4 Functions

Function Name	Description
SetContext(Rem_Add/CREP, Service)	This function checks if there exist an entry for the inputs Rem_Add/CREP and Service in the ARL and related CRL of the DP-Master (Class 2). A) If there exist an entry than it returns TRUE and sets the local context according to the current CREP and AREP. B) Otherwise it returns FALSE.
IsLastWaitingReset()	This function returns TRUE if all Reset service confirmation are received otherwise it returns FALSE.
StoreNumberIssuedResets()	This function checks stores the number of issued Reset service request in a local variable.
IsLastWaitingMinit()	This function returns TRUE if all Minit service confirmation are received otherwise it returns FALSE.
StoreNumberIssuedMinit()	This function checks stores the number of issued Minit service request in a local variable.

6.6 Application Relationship Protocol Machines (ARPMs)

6.6.1 MSCY1S

6.6.1.1 Primitive Definitions

6.6.1.1.1 Primitives Exchanged between MSCY1S and FSPMS

Table 384 - Primitives issued by FSPMS to MSCY1S

Primitive Name	Source	Associated Parameters	Functions
SInit MS0.req	FSPMS	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMS	(none)	
Abort.req	FSPMS	(none)	
Application Ready.req	FSPMS	(none)	
Check User Prm Result.req	FSPMS	Prm_OK	
Check Cfg Result.req	FSPMS	Cfg_OK, Input Data Len, Output Data Len	
Set Cfg.req	FSPMS	Cfg Data	
Set Slave Diag.req	FSPMS	Ext Diag Flag, Ext Diag Overflow, Ext Diag Data	
Set Input.req	FSPMS	Input Data	
Get Output.req	FSPMS	(none)	

Table 385 - Primitives issued by MSCY1S to FSPMS

Primitive Name	Source	Associated Parameters	Functions
SInit MS0.cnf	MSCY1S	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MSCY1S	(none)	
Check User Prm Result.cnf(+)	MSCY1S	(none)	
Check User Prm Result.cnf(-)	MSCY1S	Status	
Check Cfg Result.cnf(+)	MSCY1S	(none)	
Check Cfg Result.cnf(-)	MSCY1S	Status	
Set Cfg.cnf(+)	MSCY1S	(none)	
Set Cfg.cnf(-)	MSCY1S	(none)	
Set Slave Diag.cnf(+)	MSCY1S	(none)	
Set Slave Diag.cnf(-)	MSCY1S	(none)	
Set Input.cnf(+)	MSCY1S	(none)	
Set Input.cnf(-)	MSCY1S	(none)	
Get Output.cnf(+)	MSCY1S	Output Data, Clear Flag, New Flag	
Get Output.cnf(-)	MSCY1S	(none)	
Start.ind	MSCY1S	Actual Enabled Alarms, Alarm Sequence, Alarm Limit	
Stop.ind	MSCY1S	(none)	
Fault.ind	MSCY1S	(none)	
Set Slave Add.ind	MSCY1S	New Slave Add, Ident Number, No Add Chg, Rem Slave Data	
Check User Prm.ind	MSCY1S	User Prm Data	
Check Cfg.ind	MSCY1S	Check Cfg Mode, Cfg Data	
New Output.ind	MSCY1S	Clear Flag	
Global Control.ind	MSCY1S	Clear Command, Sync Command, Freeze Command, Group Select	

6.6.1.2 State Machine Description

The MSCY1S State Machine handles the following services issued by the DP-Master (Class 1):

diagnostic,
parameterisation,
configuration,
(cyclic) data exchange.

The MSCY1S State Machine starts (Start.req/cnf) and stops (Stop.req/cnf) the MSAC1S State Machine that handles acyclic services. The MSAC1S State Machine sends an Abort.ind to the MSCY1S State Machine that closes also the cyclic connection. The MSCY1S State Machine informs the FSPMS that the MSAC1S and the MSCY1S have been started (Start.ind) or stopped (Stop.ind).

If a Slave does not implement the parameterization structures DPV1_Status 1-3, the MSAL1S and MSAC1S State Machines, the following macros should be left empty. In this case the variable Operation_Mode remains in state V0:

CHECK_ALARM_STATE,
STOP_ASM,
SET_OPERATION_MODE,

SET_ALARM_CHKCFGM,
START_C1,
START_C1_CON,
STOP_C1_CON.

OPERATION_MODE_OK shall always return TRUE

The State Machine needs some local variables which are described below. Furthermore the machine uses macros which are listed in subclause 6.6.1.4.

Local variables

B Sync

(Octet-String)

Intermediate storage for outputs if the DP-Slave is operated in Sync Mode

B Output

(Octet-String)

Output data of the DP-Slave that can be fetched by the Get Output service. The requirements for the consistency as described in the Cfg_Data shall be taken into consideration.

B Freeze

(Octet-String)

Intermediate storage for input data.

B Input

(Octet-String)

Depending on the type (see description of Cfg_Data), the data shall be consistently loaded by the Set Input service, or may be captured freely.

B Real_Cfg

(Octet-String)

The configuration which was previously defined for the device, or the configuration which is determined in the start-up by the device, respectively. The structure of the individual octets corresponds to the structure of Cfg_Data (→Chk_Cfg).

B User Prm

(Octet-String)

Intermediate storage for User Prm data.

B Cfg

(Octet-String)

Intermediate storage for Cfg data.

Diag

(Octet-String)

Intermediate storage for Diag data.

Ext Diag Data

(Octet-String)

Intermediate storage for Ext Diag Data, the diagnosis setup by the user.

Ext Diag Flag

(Boolean)

Intermediate storage for Ext Diag Flag, which indicates important user diagnosis.

Clear Command

Sync Command

Freeze Command

Intermediate storage for global control commands.

Station_Address

(Unsigned8)

Own station address which may be set by the service "Set_Slave_Add" or which is locally stored.

Output Data Len

(Unsigned8)

Indicates the number of output bytes that are actually used.

Input Data Len

(Unsigned8)

Indicates the number of input bytes that are actually used.

Real_No_Add_Chg

(Boolean)

Indicates if an address change is possible (FALSE) or not. Can be statically defined or stored.

Active_Groups

(Unsigned8)

Contains the Group_Ident as set via Set_Prm. This is used at Global_Control for the decision if the station is addressed.

Diag_Flag

(Boolean)

This flag indicates that the Master shall be informed by a high prior Reply-Update that a change in the diagnostic information occurred.

Sync_Supp

(Boolean)

Indicates that the DP-Slave does support the Sync-Mode.

Freeze_Supp

(Boolean)

Indicates that the DP-Slave does support the Freeze-Mode.

Operation_Mode

(Boolean)

Local flag that stores the Operation_Mode set with the Set_Prm service. Operation_Mode=V0 means that the Slave does not support acyclic services on MSAC1 and alarms.

Prm_Pending

(Unsigned8)

Local flag that indicates that the User is still processing the last call of the function "Check UserPrm".

Cfg_Pending

(Unsigned8)

Local flag that indicates that the User is still processing the last call of the function "Check Cfg".

Input_Pending

(Unsigned8)

Local flag that indicates that the User has not yet set the Input data

New Output

(Unsigned8)

Local flag that indicates that there are Output data available not yet fetched with "Get Output".

DX_Entered

(Boolean)

Local flag which is set after the MSCY1S State Machine has entered the DATA-EXCH mode and at least one Data_Exchange service has been completed.

Start_State

(Unsigned8)

Local variable which is used to store the state of the MSAC1S-State Machine.

Possible values for Start_State:

Idle	MSAC1S is closed
Run	MSAC1S is open
B	Start_Operation is initiated
E	Stop_Operation is initiated
BE	Start_Operation is initiated, Stop_Operation is queued
EB	Stop_Operation is initiated, Start_Operation is queued
BEB	Start_Operation is initiated, Stop_Operation (first) and Start_Operation(second) are queued

Fail_Safe

(Boolean)

Indicates whether the DP-Slave enters the safe state when receiving a data message with the length 0 (TRUE) or data messages with zero values (FALSE).

Safe_State

(Boolean)

Indicates whether the outputs has to be switched to the safe state.

Chk Cfg Mode

(Boolean)

Indicates how to check the Cfg Data.

6.6.1.3 MSCY1S State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MSCY1S SInit MS0.req () =>Diag.Ext_Diag_Data := NIL Diag.Ext Diag Flag := FALSE B Real Cfg := Real Cfg Data Chk Cfg Mode := FALSE Sync Supp := Sync Supported Freeze Supp := Freeze Supported Diag.Ident Number := Ident Number Real No Add Chg := No Add Chg DMPMS Slave Init.req	WAIT- INI-CON
2	any state	MSCY1S Reset.req () =>MSCY1S Reset.cnf()	POWER- -ON
3	POWER-ON	MSCY1S Abort.req () =>ignore	POWER- -ON
4	POWER-ON	MSCY1S Set Slave Diag.req (Ext Diag Flag, Ext Diag Overflow, Ext Diag Data) =>MSCY1S Set Slave Diag.cnf(-)	POWER- -ON
5	POWER-ON	MSCY1S Set Cfg.req (Cfg Data) =>MSCY1S Set Cfg.cnf(-)	POWER- -ON
6	POWER-ON	MSCY1S Get Output.req () =>MSCY1S Get Output.cnf(-)	POWER- -ON
7	POWER-ON	MSCY1S Set Input.req (Input Data) =>MSCY1S Set Input.cnf(-)	POWER- -ON
8	POWER-ON	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) =>Status := Reset MSCY1S Check Cfg Result.cnf(-) (Status)	POWER- -ON
9	POWER-ON	MSCY1S Application Ready.req() =>ignore	POWER- -ON
10	POWER-ON	MSCY1S Check User Prm Result.req (Prm_OK) =>Status := Reset MSCY1S Check User Prm Result.cnf(-) (Status)	POWER- -ON
11	WAIT- INI-CON	DMPMS Slave Init.cnf () =>Operation Mode := V0 Start_State := Idle Safe_State:=TRUE Diag.Prm_Req := TRUE Diag.Cfg_Fault := FALSE Diag.Prm_Fault := FALSE Diag.Not_Supported := FALSE Diag.Master_Add := Invalid Diag.WD_On := FALSE Diag.Station_Not_Ready := TRUE Diag.Sync_Mode := FALSE Diag.Freeze_Mode := FALSE Diag.Stat_Diag := FALSE B User Prm := NIL B Cfg := NIL Diag_Flag := FALSE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) DMPMS Get Cfg Upd.req (Cfg Data) MSCY1S SInit MS0.cnf(+) ()	WAIT- PRM
12	WAIT- PRM	MSCY1S Abort.req () =>ignore	WAIT- PRM
13	WAIT- PRM	MSCY1S Set Slave Diag.req (Ext Diag Flag, Ext Diag Overflow, Ext Diag Data) =>Diag.Ext Diag Data := Ext Diag Data Diag.Ext Diag Flag := Ext Diag Flag Diag.Ext Diag Overflow := Ext Diag Overflow Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Set Slave Diag.cnf(+) ()	WAIT- PRM
14	WAIT- PRM	MSCY1S Set Cfg.req (Cfg Data) =>DMPMS Get Cfg Upd.req (Cfg Data) MSCY1S Set Cfg.cnf(+) ()	WAIT- PRM
15	WAIT- PRM	MSCY1S Get Output.req () =>MSCY1S Get Output.cnf(-) ()	WAIT- PRM

#	Current State	Event /Condition =>Actions taken&&Output	Next State
16	WAIT-PRM	MSCY1S Set Input.req (Input Data) =>MSCY1S Set Input.cnf(-) ()	WAIT-PRM
17	WAIT-PRM	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 0 =>Status := Not pending MSCY1S Check User Prm Result.cnf(-) (Status)	WAIT-PRM
18	WAIT-PRM	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 2 =>Status := New Prm Prm_Pending := 1 User Prm Data := B User Prm MSCY1S Check User Prm Result.cnf(-) (Status) MSCY1S Check User Prm.ind (User Prm Data)	WAIT-PRM
19	WAIT-PRM	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 1 && Prm_OK = TRUE =>Prm_Pending := 0 MSCY1S Check User Prm Result.cnf(+)()	WAIT-PRM
20	WAIT-PRM	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 1 && Prm_OK = FALSE =>Prm_Pending := 0 Diag.Prm_Fault := TRUE Diag Data := Diag MSCY1S Check User Prm Result.cnf(+)() DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
21	WAIT-PRM	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 0 =>Status := Not pending MSCY1S Check Cfg Result.cnf(-) (Status)	WAIT-PRM
22	WAIT-PRM	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 2 =>Status := New Cfg Cfg_Pending := 1 Cfg Data := B Cfg MSCY1S Check Cfg Result.cnf(-) (Status) MSCY1S Check Cfg.ind (Cfg Data)	WAIT-PRM
23	WAIT-PRM	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = TRUE =>Cfg_Pending := 0 MSCY1S Check Cfg Result.cnf(+)()	WAIT-PRM
24	WAIT-PRM	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = FALSE =>Cfg_Pending := 0 Diag.Cfg_Fault := TRUE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Check Cfg Result.cnf(+)()	WAIT-PRM
25	WAIT-PRM	MSCY1S Application Ready.req() =>ignore	WAIT-PRM
26	WAIT-PRM	MSAC1S Abort.ind() =>ignore	WAIT-PRM
27	WAIT-PRM	MSAC1S Start.cnf() =>START_C1_CON	WAIT-PRM
28	WAIT-PRM	MSAC1S Stop.cnf() =>Start_State := Idle	WAIT-PRM
29	WAIT-PRM	DMPMS Set Slave Add.ind(New Slave Add, Ident Number, No Add Chg, Rem Slave Data) /Real_No_Add_Chg = FALSE && Diag.Ident_Number = Ident_Number && New_Slave_Address <= 125 =>MSCY1S Set Slave Add.ind (New Slave Add, Ident Number, No Add Chg, Rem Slave Data)	POWER-ON
30	WAIT-PRM	DMPMS Set Slave Add.ind(New Slave Add, Ident Number, No Add Chg, Rem Slave Data) /Real_No_Add_Chg = TRUE Diag.Ident_Number <> Ident_Number New_Slave_Add > 125 =>ignore	WAIT-PRM
31	WAIT-PRM	DMPMS Slave Diag.ind(Req Add) =>ignore	WAIT-PRM

#	Current State	Event /Condition =>Actions taken&&Output	Next State
32	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = FALSE && Unlock_Req = FALSE =>DMPMS Set minTsdr.req (MinTsdr)	WAIT-PRM
33	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = TRUE =>ignore	WAIT-PRM
34	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = TRUE && Unlock_Req = FALSE && (Ident_Number <> Diag.Ident_Number OPERATION_MODE_OK = FALSE WD_On = TRUE && (WD_Fact_1=0 WD_Fact_2=0)) =>Diag.Prm_Fault := TRUE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
35	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = TRUE && Unlock_Req = FALSE && Ident_Number = Diag.Ident_Number && OPERATION_MODE_OK && (WD_On = FALSE (WD_Fact_1 > 0 && WD_Fact_2 > 0)) && (Freeze_Req = TRUE && Freeze Supp = FALSE Sync_Req = TRUE && Sync Supp = FALSE Prm_Data[1].0, .1, .2 = TRUE) =>Diag.Not_Supported := TRUE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
36	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len < 7 =>ignore	WAIT-PRM
37	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /PRM_OK && Prm_Pending = 0 =>Diag.Master_Add := Req Add SET_OPERATION_MODE SET_WD Active_Groups := Group_Ident Diag.Prm_Fault := FALSE Diag.Prm_Req := FALSE Diag.Not_Supported := FALSE Prm_Pending := 1 SET_ALARM_CHKCFG Input_Pending := FALSE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) DMPMS Set minTsdr.req (minTsdr) MSCY1S Check User Prm.ind (Prm Data)	WAIT-CFG
38	WAIT-PRM	DMPMS Set Prm.ind(Req Add, Prm Data) /PRM_OK && Prm_Pending > 0 =>SET_OPERATION_MODE SET_WD Active_Groups := Group_Ident Diag.Prm_Fault := FALSE Diag.Prm_Req := FALSE Diag.Not_Supported := FALSE Prm_Pending := 2 B User Prm := Prm Data.User Prm SET_ALARM_CHKCFG Input_Pending := FALSE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) DMPMS Set minTsdr.req (minTsdr)	WAIT-CFG
39	WAIT-PRM	DMPMS Chk Cfg.ind(Req Add, Cfg Data) =>ignore	WAIT-PRM
40	WAIT-CFG	MSCY1S Abort.req () =>Operation_Mode := V0 Diag.Master_Add := invalid Diag.Prm_Req := TRUE Diag.Station_Not_Ready := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM

#	Current State	Event /Condition =>Actions taken&&Output	Next State
41	WAIT-CFG	MSCY1S Set Slave Diag.req (Ext Diag Flag, Ext Diag Overflow, Ext Diag Data) =>Diag.Ext Diag Data := Ext Diag Data Diag.Ext Diag Flag := Ext Diag Flag Diag.Ext Diag Overflow := Ext Diag Overflow Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Set Slave Diag.cnf(+) ()	WAIT-CFG
42	WAIT-CFG	MSCY1S Set Cfg.req (Cfg Data) =>DMPMS Get Cfg Upd.req (Cfg Data) MSCY1S Set Cfg.cnf(+) ()	WAIT-CFG
43	WAIT-CFG	MSCY1S Get Output.req () =>MSCY1S Get Output.cnf(-) ()	WAIT-CFG
44	WAIT-CFG	MSCY1S Set Input.req (Input Data) /Input_Pending = FALSE =>MSCY1S Set Input.cnf(-) ()	WAIT-CFG
45	WAIT-CFG	MSCY1S Set Input.req (Input Data) /Input_Pending = TRUE && Operation_Mode = V0 =>Inp Data := Input Data Diag.Cfg_Fault := FALSE Diag_Flag := TRUE Diag.Stat_Diag := TRUE Diag.Station_Not_Ready := FALSE DX_Entered := FALSE Diag Data := Diag MSCY1S Set Input.cnf(+) () DMPMS Enter.req (Master Add, Input Data Len, Output Data Len) DMPMS RD Inp Upd.req (Inp Data) DMPMS Data Exchange Upd.req (Diag Flag, Inp Data) DMPMS Slave Diag Upd.req (Diag Data)	DATA-EXCH
46	WAIT-CFG	MSCY1S Set Input.req (Input Data) /Input_Pending = TRUE && Operation_Mode = V1 =>B Input := Input Data START_C1 MSCY1S Set Input.cnf(+) ()	WAIT-CFG
47	WAIT-CFG	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 0 =>Status := Not pending MSCY1S Check User Prm Result.cnf(-) (Status)	WAIT-CFG
48	WAIT-CFG	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 2 =>Status := New Prm Prm_Pending := 1 User Prm Data := User Prm Buffer MSCY1S Check User Prm Result.cnf(-) (Status) MSCY1S Check User Prm.ind (User Prm Data)	WAIT-CFG
49	WAIT-CFG	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 1 && Prm_OK = TRUE =>Prm_Pending := 0 MSCY1S Check User Prm Result.cnf(+)()	WAIT-CFG
50	WAIT-CFG	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 1 && Prm_OK = FALSE =>Prm_Pending := 0 Diag.Prm_Fault := TRUE Diag.Master_Add := Invalid Diag.Prm_Req := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag MSCY1S Check User Prm Result.cnf(+)()	WAIT-PRM
51	WAIT-CFG	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 0 =>Status := Not pending MSCY1S Check Cfg Result.cnf(-) (Status)	WAIT-CFG

#	Current State	Event /Condition =>Actions taken&&Output	Next State
52	WAIT-CFG	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 2 =>Status := New Cfg Cfg_Pending := 1 Cfg Data := B Cfg MSCY1S Check Cfg Result.cnf(-) (Status) MSCY1S Check Cfg.ind (Cfg Data)	WAIT-CFG
53	WAIT-CFG	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = TRUE && Input Data Len > 0 =>Inp Data Len := Input Data Len Outp Data Len := Output Data Len Cfg_Pending := 0 Input Pending := TRUE MSCY1S Check Cfg Result.cnf(+) ()	WAIT-CFG
54	WAIT-CFG	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = TRUE && Operation_Mode = V1 && Input Data Len = 0 =>Inp Data Len := 0 Outp Data Len := Output Data Len Inp Data := NIL Cfg_Pending := 0 START_C1 MSCY1S Check Cfg Result.cnf(+) ()	WAIT-CFG
55	WAIT-CFG	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = TRUE && Operation_Mode = V0 && Input Data Len = 0 =>Inp Data Len := 0 Outp Data Len := Output Data Len Inp Data := NIL Cfg_Pending := 0 Diag.Cfg_Fault := FALSE Diag_Flag := TRUE Diag.Stat_Diag := TRUE Diag.Station_Not_Ready := FALSE DX_Entered := FALSE Diag Data := Diag DMPMS Enter.req DMPMS Data Exchange Upd.req (Diag_Flag, Inp Data) DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Check Cfg Result.cnf(+) ()	DATA-EXCH
56	WAIT-CFG	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = FALSE =>Cfg_Pending := 0 Diag.Cfg_Fault := TRUE Diag.Prm_Req := TRUE Diag.Master_Add := Invalid StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Check Cfg Result.cnf(+) ()	WAIT-PRM
57	WAIT-CFG	MSCY1S Application Ready.req() =>ignore	WAIT-CFG
58	WAIT-CFG	MSAC1S Abort.ind() =>ignore	WAIT-CFG
59	WAIT-CFG	MSAC1S Start.cnf() /Start_State <> B =>START_C1_CON	WAIT-CFG

#	Current State	Event /Condition =>Actions taken&&Output	Next State
60	WAIT-CFG	MSAC1S Start.cnf() /Start_State = B =>Diag.Cfg_Fault := FALSE Diag_Flag := TRUE Diag.Stat_Diag := TRUE Diag.Station_Not_Ready := FALSE DX_Entered := FALSE START_C1_CON Diag Data := Diag DMPMS Enter.req DMPMS RD Inp Upd.req (Inp Data) DMPMS Data Exchange Upd.req (Diag Flag, Inp Data) DMPMS Slave Diag Upd.req (Diag Data)	DATA-EXCH
61	WAIT-CFG	MSAC1S Stop.cnf() =>STOP_C1_CON	WAIT-CFG
62	WAIT-CFG	MSAC1S Abort.ind() =>Operation_Mode := V0 Diag.Master_Add := invalid Diag.Prm_Req := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
63	WAIT-CFG	DMPMS Set Slave Add.ind(New Slave Add, Ident Number, No Add Chg, Rem Slave Data) =>ignore	WAIT-CFG
64	WAIT-CFG	DMPMS Slave Diag.ind(Req Add) /Diag.Master_Add = Req_Add =>TRIG_WD	WAIT-CFG
65	WAIT-CFG	DMPMS Slave Diag.ind(Req Add) /Diag.Master_Add <> Req_Add =>ignore	WAIT-CFG
66	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = TRUE && Diag.Master_Add = Req_Add =>Diag.Master_Add := invalid Diag.Prm_Req := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
67	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = TRUE && Diag.Master_Add <> Req_Add =>ignore	WAIT-CFG
68	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = FALSE && Lock_Req = TRUE && Diag.Master_Add = Req_Add && (Ident_Number <> Diag.Ident_Number OPERATION_MODE_OK = FALSE WD_On = TRUE && (WD_Fact_1=0 WD_Fact_2=0)) =>Diag.Prm_Fault := TRUE Diag.Master_Add := Invalid Diag.Prm_Req := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM

#	Current State	Event /Condition =>Actions taken&&Output	Next State
69	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = FALSE && Lock_Req = TRUE && Diag.Master_Add = Req_Add && Ident_Number = Diag.Ident_Number && OPERATION_MODE_OK && (WD_On = FALSE WD_Fact_1>0 && WD_Fact_2>0) && (Freeze_Req = TRUE && Freeze Supp = FALSE Sync_Req = TRUE && Sync Supp = FALSE Prm_Data[1].0, .1, .2 = TRUE) =>Diag.Not_Supported := TRUE Diag.Master_Add := Invalid Diag.Prm_Req := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
70	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = FALSE && Lock_Req = TRUE && Diag.Master_Add <> Req_Add && (Ident_Number <> Diag.Ident_Number OPERATION_MODE_OK = FALSE WD_On = TRUE && (WD_Fact_1=0 WD_Fact_2=0) Freeze_Req = TRUE && Freeze Supp = FALSE Sync_Req = TRUE && Sync Supp = FALSE Prm_Data[1].0, .1, .2 = TRUE) =>ignore	WAIT-CFG
71	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /PRM_OK && Prm_Pending = 0 =>Master_Add := Req_Add SET_OPERATION_MODE SET_WD Active_Groups := Group_Ident Prm_Pending := 1 SET_ALARM_CHKCFG Input_Pending := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) DMPMS Set minTsdr.req (minTsdr)	WAIT-CFG
72	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /PRM_OK && Prm_Pending > 0 =>Master_Add := Req_Add SET_OPERATION_MODE SET_WD Active_Groups := Group_Ident Prm_Pending := 2 B_User Prm := Prm_Data.User Prm SET_ALARM_CHKCFG Input_Pending := FALSE Stop_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) DMPMS Set minTsdr.req (minTsdr)	WAIT-CFG
73	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = FALSE && Lock_Req = FALSE =>DMPMS Set minTsdr.req (MinTsdr)	WAIT-CFG
74	WAIT-CFG	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len < 7 =>ignore	WAIT-CFG
75	WAIT-CFG	DMPMS Chk Cfg.ind(Req Add, Cfg Data) /Diag.Master_Add <> Req_Add =>ignore	WAIT-CFG
76	WAIT-CFG	DMPMS Chk Cfg.ind(Req Add, Cfg Data) /Diag.Master_Add = Req_Add && Cfg_Pending = 0 =>Cfg_Pending := 1 TRIG_WD MSCY1S Check Cfg.ind (Check Cfg Mode, Cfg Data)	WAIT-CFG

#	Current State	Event /Condition =>Actions taken&&Output	Next State
77	WAIT-CFG	DMPMS Chk Cfg.ind(Req Add, Cfg Data) /Diag.Master_Add = Req_Add && Cfg_Pending > 0 =>Cfg_Pending := 2 B Cfg := Cfg Data TRIG_WD	WAIT-CFG
78	WAIT-CFG	WDTimer expired =>Diag.Master_Add := invalid Diag.Prm_Req := TRUE StopTimer(WD) Diag.WD_On := FALSE STOP_C1 Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
79	DATA-EXCH	MSCY1S Abort.req () =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
80	DATA-EXCH	MSCY1S Set Slave Diag.req (Ext Diag Flag, Ext Diag Overflow, Ext Diag Data) =>Diag.Ext Diag Data := Ext Diag Data Diag.Ext Diag Flag := Ext Diag Flag Diag.Ext Diag Overflow := Ext Diag Overflow Diag Data := Diag Diag Flag := TRUE DMPMS Slave Diag Upd.req(Diag Data) DMPMS Data Exchange Upd.req(Diag Flag, Inp Data)	DATA-EXCH
81	DATA-EXCH	MSCY1S Set Cfg.req (Cfg Data) =>Diag.Cfg_Fault:=TRUE LEAVE-MASTER DMPMS Get Cfg Upd.req (Cfg Data) MSCY1S Set Cfg.cnf(+) () DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
82	DATA-EXCH	MSCY1S Get Output.req () /DX_Entered = TRUE =>Clear Flag := Safe_State New Flag := New Output New Output := FALSE Outp Data, Output Data := B Output DMPMS RD Outp Upd.req (Outp Data) MSCY1S Get Output.cnf(+) (Output Data, Clear Flag, New Flag)	DATA-EXCH
83	DATA-EXCH	MSCY1S Get Output.req () /DX_Entered = FALSE =>MSCY1S Get Output.cnf(-) ()	DATA-EXCH
84	DATA-EXCH	MSCY1S Set Input.req (Input Data) /Diag.Freeze_Mode = FALSE =>Inp Data, B Input := Input Data DMPMS Data Exchange Upd.req (Diag Flag, Inp Data) DMPMS RD Inp Upd.req (Inp Data) MSCY1S Set Input.cnf(+) ()	DATA-EXCH
85	DATA-EXCH	MSCY1S Set Input.req (Input Data) /Diag.Freeze_Mode = TRUE =>B Freeze := Input Data MSCY1S Set Input.cnf(+) ()	DATA-EXCH
86	DATA-EXCH	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 0 =>Status := Not pending MSCY1S Check User Prm Result.cnf(-) (Status)	DATA-EXCH
87	DATA-EXCH	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 2 =>Status := New Prm Prm_Pending := 1 User Prm Data := B User Prm MSCY1S Check User Prm Result.cnf(-) (Status) MSCY1S Check User Prm.ind (User Prm Data)	DATA-EXCH
88	DATA-EXCH	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 1 && Prm_OK = TRUE =>Prm_Pending := 0 MSCY1S Check User Prm Result.cnf(+)()	DATA-EXCH

#	Current State	Event /Condition =>Actions taken&&Output	Next State
89	DATA-EXCH	MSCY1S Check User Prm Result.req (Prm_OK) /Prm_Pending = 1 && Prm_OK = FALSE =>Diag.Prm_Fault := TRUE LEAVE-MASTER MSCY1S Check User Prm Result.cnf(+)() DMPMS Slave Diag Upd.req (Diag Data)	DATA-EXCH
90	DATA-EXCH	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 0 =>Status := Not pending MSCY1S Check Cfg Result.cnf(-) (Status)	DATA-EXCH
91	DATA-EXCH	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 2 =>Status := New Cfg Cfg_Pending := 1 Cfg Data := B Cfg MSCY1S Check Cfg Result.cnf(-) (Status) MSCY1S Check Cfg.ind (Cfg Data)	DATA-EXCH
92	DATA-EXCH	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = TRUE && Input Data Len = Inp Data Len && Output Data Len = Outp Data Len =>Cfg_Pending := 0 MSCY1S Check Cfg Result.cnf(+)()	DATA-EXCH
93	DATA-EXCH	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = TRUE && Input Data Len <> Inp Data Len && Output Data Len <> Outp Data Len =>Diag.Cfg_Fault := TRUE LEAVE-MASTER MSCY1S Check Cfg Result.cnf(+)() DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
94	DATA-EXCH	MSCY1S Check Cfg Result.req (Cfg_OK, Input Data Len, Output Data Len) /Cfg_Pending = 1 && Cfg_OK = FALSE =>Diag.Cfg_Fault := TRUE LEAVE-MASTER MSCY1S Check Cfg Result.cnf(+)() DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
95	DATA-EXCH	MSCY1S Application Ready.req() /Diag.Stat_Diag = TRUE =>Diag.Stat_Diag := FALSE Diag_Data := Diag DMPMS Slave Diag Upd.req (Diag Data) DMPMS Data Exchange Upd.req (Diag Flag, Inp Data)	DATA-EXCH
96	DATA-EXCH	MSCY1S Application Ready.req() /Diag.Stat_Diag = FALSE =>ignore	DATA-EXCH
97	DATA-EXCH	MSAC1S Abort.ind() =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
98	DATA-EXCH	MSAC1S Start.cnf() =>MSCY1S Fault.ind ()	POWER-ON
99	DATA-EXCH	MSAC1S Stop.cnf() =>MSCY1S Fault.ind ()	POWER-ON
100	DATA-EXCH	DMPMS Set Slave Add.ind(New Slave Add, Ident Number, No Add Chg, Rem Slave Data) =>ignore	DATA-EXCH
101	DATA-EXCH	DMPMS Slave Diag.ind(Req Add) /Diag.Master_Add = Req_Add && Diag.Stat_Diag = TRUE =>TRIG_WD	DATA-EXCH
102	DATA-EXCH	DMPMS Slave Diag.ind(Req Add) /Diag.Master_Add = Req_Add && Diag.Stat_Diag = FALSE =>Diag Flag := FALSE TRIG_WD DMPMS Data Exchange Upd.req (Diag Flag, Inp Data) DMPMS RD Inp Upd.req (Inp Data) MSCY1S Set Slave Diag.cnf(+)()	DATA-EXCH
103	DATA-EXCH	DMPMS Slave Diag.ind(Req Add) /Diag.Master_Add <> Req_Add =>ignore	DATA-EXCH

#	Current State	Event /Condition =>Actions taken&&Output	Next State
104	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = FALSE && Unlock_Req = FALSE =>DMPMS Set minTsdrr.req (MinTsdrr)	DATA-EXCH
105	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = TRUE && Diag.Master_Add <> Req_Add =>ignore	DATA-EXCH
106	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Unlock_Req = TRUE && Diag.Master_Add = Req_Add =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
107	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = TRUE && Unlock_Req = FALSE && Diag.Master_Add = Req_Add && (Ident_Number <> Diag.Ident_Number OPERATION_MODE_OK = FALSE WD_On = TRUE && (WD_Fact_1=0 WD_Fact_2=0)) =>Diag.Prm_Fault := TRUE LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
108	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = TRUE && Unlock_Req = FALSE && Diag.Master_Add = Req_Add && Ident_Number = Diag.Ident_Number && OPERATION_MODE_OK && (WD_On = FALSE WD_Fact_1>0 && WD_Fact_2>0) && (Freeze_Req = TRUE && Freeze_Supp = FALSE Sync_Req = TRUE && Sync_Supp = FALSE Prm_Data[1].0, .1, .2 = TRUE) =>Diag.Not_Supported := TRUE LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
109	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len >= 7 && Lock_Req = TRUE && Unlock_Req = FALSE && Diag.Master_Add <> Req_Add && (Ident_Number <> Diag.Ident_Number OPERATION_MODE_OK = FALSE WD_On = TRUE && (WD_Fact_1=0 WD_Fact_2=0) Freeze_Req = TRUE && Freeze_Supp = FALSE Sync_Req = TRUE && Sync_Supp = FALSE Prm_Data[1].0, .1, .2 = TRUE) =>ignore	DATA-EXCH
110	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Diag.Master_Add = Req_Add && PRM_OK && && (Operation_Mode = V1 (Prm_Data.len >= 10 && DPV1_Enable = TRUE)) =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
111	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Diag.Master_Add = Req_Add && PRM_OK && Operation_Mode = V0 && (Prm_Data.len < 10 DPV1_Enable = FALSE) && Prm_Pending = 0 =>SET_WD Active_Groups := Group_Ident Prm_Pending := 1 DMPMS Slave Diag Upd.req(Diag Data) DMPMS Set minTsdrr.req (minTsdrr) MSCY1S Check User Prm.ind (Prm Data)	DATA-EXCH
112	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Diag.Master_Add = Req_Add && PRM_OK && Operation_Mode = V0 && (Prm_Data.len < 10 DPV1_Enable = FALSE) && Prm_Pending > 0 =>SET_WD Active_Groups := Group_Ident Prm_Pending := 2 B User Prm := Prm Data.User Prm DMPMS Slave Diag Upd.req(Diag Data) DMPMS Set minTsdrr.req (minTsdrr)	DATA-EXCH

#	Current State	Event /Condition =>Actions taken&&Output	Next State
113	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Diag.Master_Add <> Req_Add && PRM_OK && Prm_Pending = 0 =>Diag.Sync_Mode := FALSE Diag.Freeze_Mode := FALSE Diag.Station_Not_Ready := TRUE Diag.Stat_Diag := FALSE SET_OPERATION_MODE SET_WD Active_Groups := Group_Ident Safe_State := TRUE Diag.Master_Add := Req_Add STOP_ASM STOP_C1 Prm_Pending := 1 SET_ALARM_CHKCFG Diag Data := Diag DMPMS Leave.req DMPMS Slave Diag Upd.req (Diag Data) DMPMS Set minTsdr.req (minTsdr) MSCY1S Check User Prm.ind (Prm Data)	WAIT-CFG
114	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Diag.Master_Add <> Req_Add && PRM_OK && Prm_Pending > 0 =>Diag.Sync_Mode := FALSE Diag.Freeze_Mode := FALSE Diag.Station_Not_Ready := TRUE Diag.Stat_Diag := FALSE SET_OPERATION_MODE SET_WD Active_Groups := Group_Ident Safe_State := TRUE Diag.Master_Add := Req_Add STOP_ASM STOP_C1 Prm_Pending := 2 B User Prm := Prm Data.User Prm SET_ALARM_CHKCFG Diag Data := Diag DMPMS Leave.req DMPMS Slave Diag Upd.req (Diag Data) DMPMS Set minTsdr.req (minTsdr)	WAIT-CFG
115	DATA-EXCH	DMPMS Set Prm.ind(Req Add, Prm Data) /Prm_Data.len < 7 =>ignore	DATA-EXCH
116	DATA-EXCH	DMPMS Chk Cfg.ind(Req Add, Cfg Data) /Diag.Master_Add <> Req_Add =>ignore	DATA-EXCH
117	DATA-EXCH	DMPMS Chk Cfg.ind(Req Add, Cfg Data) /Diag.Master_Add = Req_Add && Cfg_Pending = 0 =>Cfg_Pending := 1 TRIG_WD MSCY1S Check Cfg.ind (Check Cfg Mode, Cfg Data)	DATA-EXCH
118	DATA-EXCH	DMPMS Chk Cfg.ind(Req Add, Cfg Data) /Diag.Master_Add = Req_Add && Cfg_Pending > 0 =>Cfg_Pending := 2 B Cfg := Cfg Data TRIG_WD	DATA-EXCH
119	DATA-EXCH	DMPMS Data Exchange.ind(Outp Data) /Outp_Data.len > 0 && Outp_Data.len = Outp Data Len && Diag.Sync_Mode = FALSE =>B Output := Outp Data TRIG_WD Safe_State := FALSE Clear Flag := FALSE New Output := TRUE CHECK_ALARM_START MSCY1S New Output.ind (Clear Flag)	DATA-EXCH

#	Current State	Event /Condition =>Actions taken&&Output	Next State
120	DATA-EXCH	DMPMS Data Exchange.ind(Outp Data) /Outp_Data.len > 0 && Outp_Data.len = Outp Data Len && Diag.Sync_Mode = TRUE && Fail_Safe = FALSE =>B Sync := Outp Data TRIG_WD CHECK_ALARM_START	DATA-EXCH
121	DATA-EXCH	DMPMS Data Exchange.ind(Outp Data) /Outp_Data.len > 0 && Outp_Data.len = Outp Data Len && Diag.Sync_Mode = TRUE && Fail_Safe = TRUE =>B Sync := Outp Data TRIG_WD Safe_State := FALSE CHECK_ALARM_START	DATA-EXCH
122	DATA-EXCH	DMPMS Data Exchange.ind(Outp Data) /Outp_Data.len = 0 && Outp Data Len = 0 =>CHECK_ALARM_START	DATA-EXCH
123	DATA-EXCH	DMPMS Data Exchange.ind(Outp Data) /Outp_Data.len <> Outp Data Len && (Outp_Data.len = 0 && Fail_Safe = TRUE) =>TRIG_WD Safe_State := TRUE Clear Flag := TRUE New Output := TRUE CHECK_ALARM_START MSCY1S New Output.ind (Clear Flag)	DATA-EXCH
124	DATA-EXCH	DMPMS Data Exchange.ind(Outp Data) /Outp_Data.len <> Outp Data Len && (Outp_Data.len <= 0 Fail_Safe = FALSE) =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
125	DATA-EXCH	WDTimer expired =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
126	DATA-EXCH	DMPMS Global Control.ind(Control Command, Group Select) /Group_Select <> 0 && ((Active_Groups & Group_Select) = 0) =>ignore	DATA-EXCH
127	DATA-EXCH	DMPMS Global Control.ind(Control Command, Group Select) /(Group_Select = 0 (Active_Groups & Group_Select) <> 0) && (Control_Command.0, .6, .7 = TRUE) =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
128	DATA-EXCH	DMPMS Global Control.ind(Control Command, Group Select) /Control_Command.Clear Command = FALSE && (Group_Select = 0 (Active_Groups & Group_Select) <> 0) && (Control_Command.0, .6, .7 = FALSE) =>Clear Command := FALSE	CHECK-SYNC
129	DATA-EXCH	DMPMS Global Control.ind(Control Command, Group Select) /Control_Command.Clear Command = TRUE && (Group_Select = 0 (Active_Groups & Group_Select) <> 0) && (Control_Command.0, .6, .7 = FALSE) =>Clear Command := TRUE Clear Flag := TRUE New Output := TRUE Safe_State := TRUE	CHECK-SYNC
130	CHECK-SYNC	/NOSYNC =>Sync Command := 0	CHECK-FREEZE
131	CHECK-SYNC	/UNSYNC && Sync Supp = FALSE =>Sync Command :=1	CHECK-FREEZE
132	CHECK-SYNC	/SYNC && Sync Supp = FALSE =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM

#	Current State	Event /Condition =>Actions taken&&Output	Next State
133	CHECK-SYNC	/UNSYNC && Sync Supp = TRUE && B Sync <> Nil =>B Output := B Sync B Sync := Nil Sync Command := 1 Clear Flag := Safe_State New Output t := TRUE Diag.Sync_Mode := FALSE Diag Data := Diag MSCY1S New Output.ind (Clear Flag) DMPMS Slave Diag Upd.req(Diag Data)	CHECK-FREEZE
134	CHECK-SYNC	/UNSYNC && Sync Supp = TRUE && B Sync = Nil =>Sync Command := 1 Clear Flag := Safe_State Diag.Sync_Mode := FALSE Diag Data := Diag DMPMS Slave Diag Upd.req(Diag Data)	CHECK-FREEZE
135	CHECK-SYNC	/SYNC && Sync Supp = TRUE && Diag.Sync_Mode = FALSE =>B Sync := Nil Sync Command := 2 Diag.Sync_Mode := TRUE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data)	CHECK-FREEZE
136	CHECK-SYNC	/SYNC && Sync Supp = TRUE && Diag.Sync_Mode = TRUE && B Sync <> Nil =>B Output := B Sync B Sync := Nil Clear Flag := Safe State New Output := TRUE Sync Command:=2 MSCY1S New Output.ind (Clear Flag)	CHECK-FREEZE
137	CHECK-SYNC	/SYNC && Sync Supp = TRUE && Diag.Sync_Mode = TRUE && B Sync = Nil =>Sync Command := 2	CHECK-FREEZE
138	CHECK-FREEZE	/NOFREEZE =>Freeze Command := 0 MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH
139	CHECK-FREEZE	/UNFREEZE && Freeze Supp = FALSE =>Freeze Command := 1 MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH
140	CHECK-FREEZE	/FREEZE && Freeze Supp = FALSE =>LEAVE-MASTER DMPMS Slave Diag Upd.req (Diag Data)	WAIT-PRM
141	CHECK-FREEZE	/UNFREEZE && Freeze Supp = TRUE && B Freeze <> Nil =>Inp Data := B Freeze B Freeze := Nil Freeze Command := 1 Diag.Freeze_Mode := FALSE Diag Data := Diag DMPMS Data Exchange Upd.req (Diag Flag, Inp Data) DMPMS RD Inp Upd.req (Inp Data) DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH
142	CHECK-FREEZE	/UNFREEZE && Freeze Supp = TRUE && B Freeze = Nil =>Freeze Command := 1 Diag.Freeze_Mode := FALSE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH

#	Current State	Event /Condition =>Actions taken&&Output	Next State
143	CHECK-FREEZE	/FREEZE && Freeze Supp = TRUE && Diag.Freeze_Mode = FALSE =>B Freeze := Nil Freeze Command := 2 Diag.Freeze_Mode := FALSE Diag Data := Diag DMPMS Slave Diag Upd.req (Diag Data) MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH
144	CHECK-FREEZE	/FREEZE && Freeze Supp = TRUE && Diag.Freeze_Mode = TRUE && B Freeze <> Nil =>Inp Data := B Freeze B Freeze := Nil Freeze Command := 2 DMPMS Data Exchange Upd.req (Diag Flag, Inp Data) DMPMS RD Inp Upd.req (Inp Data) MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH
145	CHECK-FREEZE	/FREEZE && Freeze Supp = TRUE && Diag.Freeze_Mode = TRUE && B Freeze = Nil =>Freeze Command := 2 MSCY1S Global Control.ind (Clear Command, Sync Command, Freeze Command)	DATA-EXCH

6.6.1.4 Functions

Macros

LEAVE-MASTER

```
(
  Diag.Master_Add=invalid
  Diag.Sync_Mode=FALSE
  Diag.Freeze_Mode=FALSE
  Diag.Prm_Req=TRUE
  Diag.Station_Not_Ready=TRUE
  Stop Timer(WD) Diag.WD_On=FALSE
  Diag.Stat_Diag=FALSE
  Operation_Mode=V0
  Safe_State=TRUE
  STOP_ASM, STOP_C1
  DMPMS_Leave.Req
  Diag_Data=Diag
  DMPMS_Slave_Diag_Upd.req(Diag_Data)
)
```

PRM_OK

```
(
  Prm_Data.len>=7 && Lock_Req=TRUE && Unlock_Req=FALSE
  && Ident_Number=Real_Ident
  && (WD_On=FALSE ||
    (WD_On=TRUE && WD_Fact_1>0 && WD_Fact_2>0))
  && (Freeze_Req=FALSE || Freeze_Supported)
  && (Sync_Req=FALSE || Sync_Supported)
  && (Prm_Data[1].0, .1, .2=FALSE)
  && OPERATION_MODE_OK
)
```

STOP_ASM

```
(
  IF (DX_Entered=TRUE) THEN (MSCY1S_Stop.ind)
)
```

CHECK_ALARM_START

```
(
  IF (DX_Entered=FALSE && Diag.Stat_Diag=FALSE)
  THEN( DX_Entered=TRUE,
        MSCY1S_Start.ind(Actual_Enabled_Alarms, Alarm_Sequence, Alarm_Limit)
  )
```

SET_WD

```
(
  IF (WD_On=TRUE)
  THEN (Start_Timer(WD), Diag.WD_On=TRUE)
  ELSE (Stop_Timer(WD), Diag.WD_On=FALSE)
)
```

TRIG_WD

```
(
  IF (Diag.WD_On=TRUE)
  THEN Start_Timer(WD)
)
```

SET_OPERATION_MODE

```
(
  IF (Prm_Data.len ≥ 10 && DPV1_Enable=TRUE)
  THEN (Operation_Mode=V1)
  ELSE (Operation_Mode=V0)
)
```

OPERATION_MODE_OK

```
(
  Prm_Data.len < 10 ||
  ( IsARexistent(MS1) && DPV1_Enable )
)
```

SET_ALARM_CHKCFG

```
(
  IF (Prm_Data.len ≥ 10 && DPV1_Enable=TRUE)
  THEN (Enabled_Alarms=DPV1_Status_2.2 - DPV1_Status_2.7)
  ELSE (Enabled_Alarms=0)
  IF (Prm_Data.len ≥ 10)
  THEN (Alarm_Mode_Master=DPV1_Status_3.0 - DPV1_Status_3.2)
  ELSE (Alarm_Mode_Master=0)
  IF (Prm_Data.len ≥ 10)
  THEN (Alarm_Mode_Master=DPV1_Status_3.0 - DPV1_Status_3.2)
  ELSE (Chk Cfg Mode = DPV1_Status_2.0)
)
```

START_C1

```
(
  IF (Start_State=Idle)
  THEN (MSAC1S_Start.req(Diag.Master_Add), Start_State=B)
  IF (Start_State=E)
  THEN (Start_State=EB)
  IF (Start_State=BE)
  THEN (Start_State=BEB)
)
```

STOP_C1

```
(
  IF (Start_State=B)
  THEN (Start_State=BE)
  IF (Start_State=Run)
  THEN (MSAC1S_Stop.req, Start_State=E)
  IF (Start_State=EB)
  THEN (Start_State=E)
  IF (Start_State=BEB)
  THEN (Start_State=BE)
)
```

START_C1_CON

```
(
  IF (Start_State=B)
  THEN (Start_State=Run)
  IF (Start_State=BE)
  THEN (MSAC1S_Stop.req, Start_State=E)
  IF (Start_State=BEB)
  THEN (MSAC1S_Stop.req, Start_State=EB)
)
```

STOP_C1_CON

```
(
  IF (Start_State=E)
  THEN (Start_State=Idle)
  IF (Start_State=EB)
  THEN (MSAC1S_Start.req(Diag.Master_Add), Start_State=B)
)
```

6.6.2 MSAC1S**6.6.2.1 Primitive Definitions****6.6.2.1.1 Primitives Exchanged between MSAC1S and FSPMS****Table 386 - Primitives issued by FSPMS to MSAC1S**

Primitive Name	Source	Associated Parameters	Functions
SInit MS1.req	FSPMS	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMS	(none)	
Read.rsp(+)	FSPMS	Length, Data	
Read.rsp(-)	FSPMS	Error Decode, Error Code 1 Error Code 2	
Write.rsp(+)	FSPMS	Length	
Write.rsp(-)	FSPMS	Error Decode, Error Code 1 Error Code 2	
Alarm Ack.rsp(+)	FSPMS	Slot Number, Alarm Type, Seq Nr	
Alarm Ack.rsp(-)	FSPMS	(none)	

Table 387 - Primitives issued by MSAC1S to FSPMS

Primitive Name	Source	Associated Parameters	Functions
SInit MS1.cnf	MSAC1S	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MSAC1S	(none)	
Fault.ind	MSAC1S	(none)	
Read.ind	MSAC1S	Slot Number, Index, Length	
Write.ind	MSAC1S	Slot Number, Index, Length, Data	
Alarm Ack.ind	MSAC1S	Slot Number, Alarm Type, Seq Nr	

6.6.2.1.2 Primitives Exchanged between MSAC1S and MSCY1S

Table 388 - Primitives issued by MSCY1S to MSAC1S

Primitive Name	Source	Associated Parameters	Functions
Start.req	MSCY1S	Master Add	Start Processing of acyclic services
Stop.req	MSCY1S	(none)	Stop Processing of acyclic services

Table 389 - Primitives issued by MSAC1S to MSCY1S

Primitive Name	Source	Associated Parameters	Functions
Start.cnf	MSAC1S	(none)	MSAC1 is ready to execute acyclic services
Stop.cnf	MSAC1S	(none)	MSAC1 is disabled
Abort.ind	MSAC1S	(none)	Termination of MSAC1 due to a sequence error or wrong protocol elements

6.6.2.1.3 Parameter of MSAC1S Primitives

The parameters used with the primitives exchanged between the FSPMS and the MSAC1S are described in FAL Type 3 Service Definition IEC 61158-5.

Table 390 – Parameter Used with Primitives Exchanged between MSAC1S and MSCY1S

Parameter Name	Description
MasterAdd	This parameter conveys the DL address of the assigned DP-Master.

6.6.2.2 State Machine Description

After Power-On the MSAC1S waits for initiation by means of SInit MS1.

In the state CLOSED the service Start called from the MSCY1S is expected. Then the access protection for the SAP 51 (50) will be activated by means of the function RSAP_ACTIVATE.

When receiving the confirmation the State Machine enters the OPEN state. In this state the services Read, Write and Alarm_Ack are allowed. Only one request from the DP-Master is processed simultaneously on each SAP. Additionally to Read/Write at SAP 51 a Alarm_Ack may be executed at SAP 50.

When a valid service is invoked by the DP-Master, the MSAC1S will indicate this primitive to the AP-Context (Read/Write or Alarm_Ack) and waits for response. After receiving the corresponding response MSAC1S will provide this PDU in the corresponding SAP update-buffer. The service is finished after DP-Master has fetched the response.

The local service Stop called from the MSCY1S is closing the connection. The SAP 51 (50) is deactivated and a possibly stored response in the DL is deleted.

The abort of the connection is indicated to the MSCY1S with the local service Abort. If the User is still processing a response while the connection is aborted this response shall be discarded by the User.

The State Machine needs some local variables which are described below and local functions which are described in subclause 6.6.2.4.

Local variables of the MSAC1S**Server_SAP**

(Unsigned8)

This SAP (51) is used for the MSAC1S_Read/_Write and MSAC1S_Alarm_Ack services.

Alarm_SAP

(Unsigned8)

This SAP (50) is reserved for MSAC1S_Alarm_Ack services.

Res_SAP

(Unsigned8)

Variable holding the SAP for the MSAC1S_Alarm_Ack -Response.

Service_Header

(Unsigned8)

The actual service header of the service processed. Used to check whether the User generates the correct response.

AA_State

MSAC1S_Alarm_Ack services may be executed simultaneously with the MSAC1S_Read/_Write-services. Thus, the actual state of the MSAC1S_Alarm_Ack processing is stored in the variable AA_State:

Values of AA_State:

Idle	No MSAC1S_Alarm_Ack in progress
WRes	MSAC1S_Alarm_Ack.ind is sent to the MSAL1S waiting for response
Upd	Response of the MSAL1S will be sent to the Update-Buffer
SRes	Response is stored - waiting for the next poll of the DP-Master

VS_Upd

Indicates that a Response will be sent to the Update-Buffer.

6.6.2.3 MSAC1S State Table

Table 391 - MSAC1S State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MSAC1_SInit_MS1.req =>Server_SAP:=51 Alarm_SAP:=50 MSAC1_SInit_MS1.cnf	CLOSED
2	POWER-ON	MSAC1_Reset.req =>MSAC1_Reset.cnf	POWER-ON
3	POWER-ON	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) =>MSAC1_Fault.ind	POWER-ON
4	CLOSED	MSAC1S_Start.req(Master_Add) =>DMPMS_RSAP_ACTIVATE.req(SSAP=Server_SAP, Access=Master_Add, L_sdu_length_list=Nil,Indication_Mode=Unchanged)	SET-ACC51
5	CLOSED	MSAC1S_Stop.req =>MSAC1_Stop.cnf	CLOSED
6	SET-ACC51	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status = OK =>DMPMS_RSAP_ACTIVATE.req(SSAP=Alarm_SAP, Access=Master_Add, L_sdu_length_list=Nil,Indication_Mode=Unchanged)	SET-ACC50
7	SET-ACC51	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status=NO/IV =>MSAC1_Fault.ind	POWER-ON
8	SET-ACC50	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Alarm_SAP,M_status) /M_status = OK =>AA_State:=Idle VS_Upd:=False MSAC1_Start.cnf	OPEN
9	SET-ACC50	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Alarm_SAP,M_status) /M_status=NO/IV =>MSAC1_Fault.ind	POWER-ON
10	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
11	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
12	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Read-REQ-PDU =>Service_Header:=Function_Num MSAC1_Read.ind(Req_Add=Loc_add, Slot_Number,Index,Length)	VS-WRES
13	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Write-REQ-PDU =>Service_Header:=Function_Num MSAC1_Write.ind(Req_Add=Loc_add, Slot_Number,Index,Length,Data)	VS-WRES
14	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Alarm_Ack-REQ-PDU =>AA_State:=WRes Res_SAP:=Server_SAP MSAC1_Alarm_Ack.ind(Alarm_Type, Slot_Number,Seq_Nr)	AA-WRES
15	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Alarm_Ack-REQ-PDU =>AA_State:=WRes Res_SAP:=Alarm_SAP MSAC1_Alarm_Ack.ind(Alarm_Type, Slot_Number,Seq_Nr)	AA-WRES
16	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /NOT(Read-REQ-PDU Write-REQ-PDU Alarm_Ack-REQ-PDU) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51

#	Current State	Event /Condition =>Actions taken&&Output	Next State
17	OPEN	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /NOT(Alarm_Ack-REQ-PDU) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
18	OPEN	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) =>MSAC1_Fault.ind	POWER-ON
19	OPEN	MSAC1_Read.rsp(+)(Length, Data) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
20	OPEN	MSAC1_Read.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
21	OPEN	MSAC1_Write.rsp(+)(Length) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
22	OPEN	MSAC1_Write.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
23	OPEN	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
24	OPEN	MSAC1S_Stop.req =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	STOP-DEA51
25	OPEN	MSAC1S_Start.req(Master_Add) =>MSAC1_Fault.ind	POWER-ON
26	OPEN	MSAC1_SInit_MS1.req =>MSAC1_Fault.ind	POWER-ON
27	OPEN	MSAC1_Reset.req =>MSAC1_Reset.cnf	POWER-ON
28	VS-WRES	MSAC1_Read.rsp(+)(Length, Data) /Service_Header=0x5E && Max_DLSDU_Length_Req_Low>=(Length+4) =>VS_Upd:=TRUE L_sdu:=Read-RES-PDU DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low,Transmit=Single)	VS-SRES
29	VS-WRES	MSAC1_Read.rsp(+)(Length, Data) /Service_Header=0x5E && Max_DLSDU_Length_Req_Low<(Length+4) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
30	VS-WRES	MSAC1_Write.rsp(+)(Length) /Service_Header=0x5F =>VS_Upd:=True L_sdu:=Write-RES-PDU DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low,Transmit=Single)	VS-SRES
31	VS-WRES	MSAC1_Read.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) /Service_Header=0x5E =>VS_Upd:=True L_sdu:=Read-NRS-PDU DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low,Transmit=Single)	VS-SRES
32	VS-WRES	MSAC1_Write.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) /Service_Header=0x5F =>VS_Upd:=True L_sdu:=Write-NRS-PDU DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low,Transmit=Single)	VS-SRES
33	VS-WRES	MSAC1_Read.rsp(+)(Length, Data) /Service_Header<>0x5E =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
34	VS-WRES	MSAC1_Read.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) /Service_Header<>0x5E =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
35	VS-WRES	MSAC1_Write.rsp(+)(Length) /Service_Header<>0x5F =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51

#	Current State	Event /Condition =>Actions taken&&Output	Next State
36	VS-WRES	MSAC1_Write.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) /Service_Header<>0x5F =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
37	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
38	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
39	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
40	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /ALARM_ACK-REQ-PDU && AA_State=Idle =>AA_State=WRes Res_SAP:=Alarm_SAP MSAC1_Alarm_Ack.ind(Alarm_Type, Slot_Number,Seq_Nr)	VS-WRES
41	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && AA_State=SRes =>AA_State=Idle	VS-WRES
42	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /NOT((ALARM_ACK-REQ-PDU && AA_State=Idle) (Update_status=LO && L_sdu.len=0 && AA_State=SRes)) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
43	VS-WRES	MSAC1S_Start.req(Master_Add) =>MSAC1_Fault.ind	POWER-ON
44	VS-WRES	MSAC1S_Stop.req =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	STOP-DEA51
45	VS-WRES	MSAC1_Alarm_Ack.rsp(Alarm_Type, Slot_Number, Seq_Nr) /AA_State<=WRes =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
46	VS-WRES	MSAC1_Alarm_Ack.rsp(Alarm_Type, Slot_Number, Seq_Nr) /AA_State=WRes =>AA_State=Upd DMPMS_REPLY_UPDATE.req(SSAP=Res_SAP, L_sdu=Alarm_Ack-RES-PDU, Serv_class=Low,Transmit=Single)	VS-WRES
47	VS-WRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Alarm_SAP,Serv_class,L_status) /L-status=OK && AA_State=Upd =>AA_State=SRes	VS-WRES
48	VS-WRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Alarm_SAP,Serv_class,L_status) /L-status=OK && AA_State<=Upd =>MSAC1_Fault.ind	POWER-ON
49	VS-WRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L_status=LS/IV/LR =>MSAC1_Fault.ind	POWER-ON
50	VS-WRES	MSAC1_SInit_MS1.req =>MSAC1_Fault.ind	POWER-ON
51	VS-WRES	MSAC1_Reset.req =>MSAC1_Reset.cnf	POWER-ON
52	VS-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L-status=OK && VS_Upd=True =>VS_Upd=False	VS-SRES
53	VS-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L_status=LS/IV/LR =>MSAC1_Fault.ind	POWER-ON
54	VS-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L-status=OK && VS_Upd=False =>MSAC1_Fault.ind	POWER-ON

#	Current State	Event /Condition =>Actions taken&&Output	Next State
55	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
56	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
57	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && AA_State=Idle&& VS_Upd=False	OPEN
58	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && AA_State=WRes && VS_Upd=False	AA-WRES
59	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && (AA_State=SRes AA_State=Upd) && VS_Upd=False	AA-SRES
60	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && VS_Upd=True =>MSAC1_Fault.ind	POWER-ON
61	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len<>0 =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
62	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /ALARM_ACK-REQ-PDU && AA_State=Idle =>AA_State=WRes Res_SAP:=Alarm_SAP MSAC1_Alarm_Ack.ind(Alarm_Type, Slot_Number, Seq_Nr)	VS-SRES
63	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && AA_State=SRes =>AA_State=Idle	VS-SRES
64	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /NOT((ALARM_ACK-REQ-PDU && AA_State=Idle) (Update_status=LO && L_sdu.len=0 && AA_State=SRes)) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
65	VS-SRES	MSAC1S_Start.req(Master_Add) =>MSAC1_Fault.ind	POWER-ON
66	VS-SRES	MSAC1S_Stop.req =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	STOP-DEA51
67	VS-SRES	MSAC1_Read.rsp(+)(Length, Data) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
68	VS-SRES	MSAC1_Read.rsp(-)(Error Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
69	VS-SRES	MSAC1_Write.rsp(+)(Length) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
70	VS-SRES	MSAC1_Write.rsp(-)(Error Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
71	VS-SRES	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) /AA_State<>WRes =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51

#	Current State	Event /Condition =>Actions taken&&Output	Next State
72	VS-SRES	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) /AA_State=WRes =>AA_State:=Upd DMPMS_REPLY_UPDATE.req(SSAP=Res_SAP, L_sdu=Alarm_Ack-RES-PDU, Serv_class=Low,Transmit=Single)	VS-SRES
73	VS-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Alarm_SAP,Serv_class,L_status) /L-status=OK && AA_State=Upd =>AA_State:=SRes	VS-SRES
74	VS-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Alarm_SAP,Serv_class,L_status) /L-status=OK && AA_State<>Upd =>MSAC1_Fault.ind	POWER-ON
75	VS-SRES	MSAC1_SInit_MS1.req =>MSAC1_Fault.ind	POWER-ON
76	VS-SRES	MSAC1_Reset.req =>MSAC1_Reset.cnf	POWER-ON
77	AA-WRES	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) /Res_SAP=Server_SAP =>AA_State:=Upd L_sdu:=Alarm_Ack-RES-PDU DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low,Transmit=Single)	AA-SRES
78	AA-WRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) =>MSAC1_Fault.ind	POWER-ON
79	AA-WRES	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) /Res_SAP=Alarm_SAP =>AA_State:=Upd DMPMS_REPLY_UPDATE.req(SSAP=Res_SAP, L_sdu=Alarm_Ack-RES-PDU, Serv_class=Low,Transmit=Single)	AA-SRES
80	AA-WRES	MSAC1_Read.rsp(+)(Length, Data) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
81	AA-WRES	MSAC1_Read.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
82	AA-WRES	MSAC1_Write.rsp(+)(Length) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
83	AA-WRES	MSAC1_Write.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
84	AA-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
85	AA-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
86	AA-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
87	AA-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Read-REQ-PDU && Res_SAP=Alarm_SAP =>Service_Header:=Function_Num MSAC1_Read.ind(Req_Add=Loc_add, Slot_Number,Index,Length)	VS-WRES
88	AA-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Write-REQ-PDU && Res_SAP=Alarm_SAP =>Service_Header:=Function_Num MSAC1_Write.ind(Req_Add=Loc_add, Slot_Number,Index,Length,Data)	VS-WRES
89	AA-WRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /NOT(Write-REQ-PDU Read-REQ-PDU) Res_SAP=Server_SAP =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
90	AA-WRES	MSAC1S_Start.req(Master_Add) =>MSAC1_Fault.ind	POWER-ON

#	Current State	Event /Condition =>Actions taken&&Output	Next State
91	AA-WRES	MSAC1S_Stop.req =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	STOP-DEA51
92	AA-WRES	MSAC1_Reset.req =>MSAC1_Reset.cnf	POWER-ON
93	AA-WRES	MSAC1_SInit_MS1.req =>MSAC1_Fault.ind	POWER-ON
94	AA-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Alarm_SAP,Serv_class,L_status) /L-status=OK && AA_State=Upd && Res_SAP=Alarm_SAP =>AA_State:=SRes	AA-SRES
95	AA-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L_status=LS/IV/LR =>MSAC1_Fault.ind	POWER-ON
96	AA-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Alarm_SAP,Serv_class,L_status) /L-status=OK && (AA_State<>Upd Res_SAP<>Alarm_SAP) =>MSAC1_Fault.ind	POWER-ON
97	AA-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L-status=OK && AA_State=Upd && Res_SAP=Server_SAP =>AA_State:=SRes	AA-SRES
98	AA-SRES	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP,Serv_class,L_status) /L-status=OK && (AA_State<>Upd Res_SAP<>Server_SAP) =>MSAC1_Fault.ind	POWER-ON
99	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
100	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /L_sdu.len=0 && Update_status=NO =>MSAC1_Fault.ind	POWER-ON
101	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && AA_State=SRes =>AA_State:=Idle	OPEN
102	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Alarm_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status<>LO AA_State<>SRes && Res_SAP=Alarm_SAP =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
103	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status=LO && L_sdu.len=0 && AA_State=SRes =>AA_State:=Idle	OPEN
104	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Update_status<>LO L_sdu.len<>0 AA_State<>SRes && Res_SAP=Server =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
105	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Read-REQ-PDU && RES_SAP=ALARM_SAP =>Service_Header:=Function_Num MSAC1_Read.ind(Req_Add=Loc_add, Slot_Number,Index,Length)	VS-WRES
106	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /Write-REQ-PDU && RES_SAP=ALARM_SAP =>Service_Header:=Function_Num MSAC1_Write.ind(Req_Add=Loc_add, Slot_Number,Index,Length,Data)	VS-WRES
107	AA-SRES	DMPMS_DATA_REPLY.ind(SSAP,DSAP=Server_SAP,Loc_add,Rem_add,L_sdu,Serv_cl ass, Update_status) /NOT(Write-REQ-PDU Read-REQ-PDU) && RES_SAP=ALARM_SAP =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
108	AA-SRES	MSAC1S_Start.req(Master_Add) =>MSAC1_Fault.ind	POWER-ON
109	AA-SRES	MSAC1S_Stop.req =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	STOP-DEA51

#	Current State	Event /Condition =>Actions taken&&Output	Next State
110	AA-SRES	MSAC1_Read.rsp(+)(Length, Data) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
111	AA-SRES	MSAC1_Read.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
112	AA-SRES	MSAC1_Write.rsp(+)(Length) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
113	AA-SRES	MSAC1_Write.rsp(-)(Error_Decode, Error_Code_1, Error_Code_2) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
114	AA-SRES	MSAC1_Alarm_Ack.rsp (Alarm_Type, Slot_Number, Seq_Nr) =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC1S_Abort.ind	ERR-DEA51
115	AA-SRES	MSAC1_SInit_MS1.req =>MSAC1_Fault.ind	POWER-ON
116	AA-SRES	MSAC1_Reset.req =>MSAC1_Reset.cnf	POWER-ON
117	STOP-DEA51	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status = OK =>DMPMS_SAP_DEACTIVATE.req(SSAP=Alarm_SAP)	STOP-DEA50
118	STOP-DEA51	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status=NO/IV =>MSAC1_Fault.ind	POWER-ON
119	STOP-DEA50	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Alarm_SAP,M_status) /M_status = OK =>MSAC1_Stop.cnf	CLOSED
120	STOP-DEA50	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Alarm_SAP,M_status) /M_status=NO/IV =>MSAC1_Fault.ind	POWER-ON
121	ERR-DEA51	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status = OK =>DMPMS_SAP_DEACTIVATE.req(SSAP=Alarm_SAP)	ERR-DEA50
122	ERR-DEA51	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status=NO/IV =>MSAC1_Fault.ind	POWER-ON
123	ERR-DEA50	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Alarm_SAP,M_status) /M_status = OK	CLOSED
124	ERR-DEA50	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Alarm_SAP,M_status) /M_status=NO/IV =>MSAC1_Fault.ind	POWER-ON

6.6.2.4 Functions

b) Local functions of the MSAC1S

Table 392 - Function Read-REQ-PDU ()

Function Name	Description
Read-REQ-PDU ()	This function will return TRUE if the corresponding L_sdu is a valid Read-REQ-PDU.

Table 393 - Function Write-REQ-PDU ()

Function Name	Description
Write-REQ-PDU ()	This function will return TRUE if the corresponding L_sdu is a valid Write-REQ-PDU.

Table 394 - Function Alarm_Ack-REQ-PDU ()

Function Name	Description
Alarm_Ack-REQ-PDU ()	This function will return TRUE if the corresponding L_sdu is a valid Alarm_Ack-REQ-PDU.

6.6.3 MSRM2S

6.6.3.1 Primitive Definitions

6.6.3.1.1 Primitives Exchanged between MSRM2S and FSPMS

Table 395 - Primitives issued by FSPMS to MSRM2S

Primitive Name	Source	Associated Parameters	Functions
SInit MS2.req	FSPMS	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMS	(none)	

Table 396 - Primitives issued by MSRM2S to FSPMS

Primitive Name	Source	Associated Parameters	Functions
SInit MS2.cnf	MSRM2S	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MSRM2S	(none)	
Fault.ind	MSRM2S	(none)	

6.6.3.2 State Machine Description

For acyclic communication this addressing mechanism is changed to a more flexible one. The DSAPs are allocated dynamically by the so called Resource Manager (RM) of the DP-Slave.

The Resource Manager (RM) co-ordinates the communication resources for MS2 AR of the DP-Slave. The main features of the RM are described in the following sequence:

- The DP-Master requests a communication connection by means of the service Initiate (DSAP 49). If multiple applications processes are used in the DP-Slave it is possible to address these application processes through an Application Process Identifier (API).
- The RM of the DP-Slave provides an unused SAP (SAP 0 - 48). The possible SAPs are defined locally in the Res_SAP_List.
- The RM responds the Initiate.req immediately with the RM_REQ-PDU containing the corresponding unused SAP.
- Further steps following the Initiate.req (Initiate.res, Read, Write, Abort) are handled by the corresponding MSAC2S State Machine.

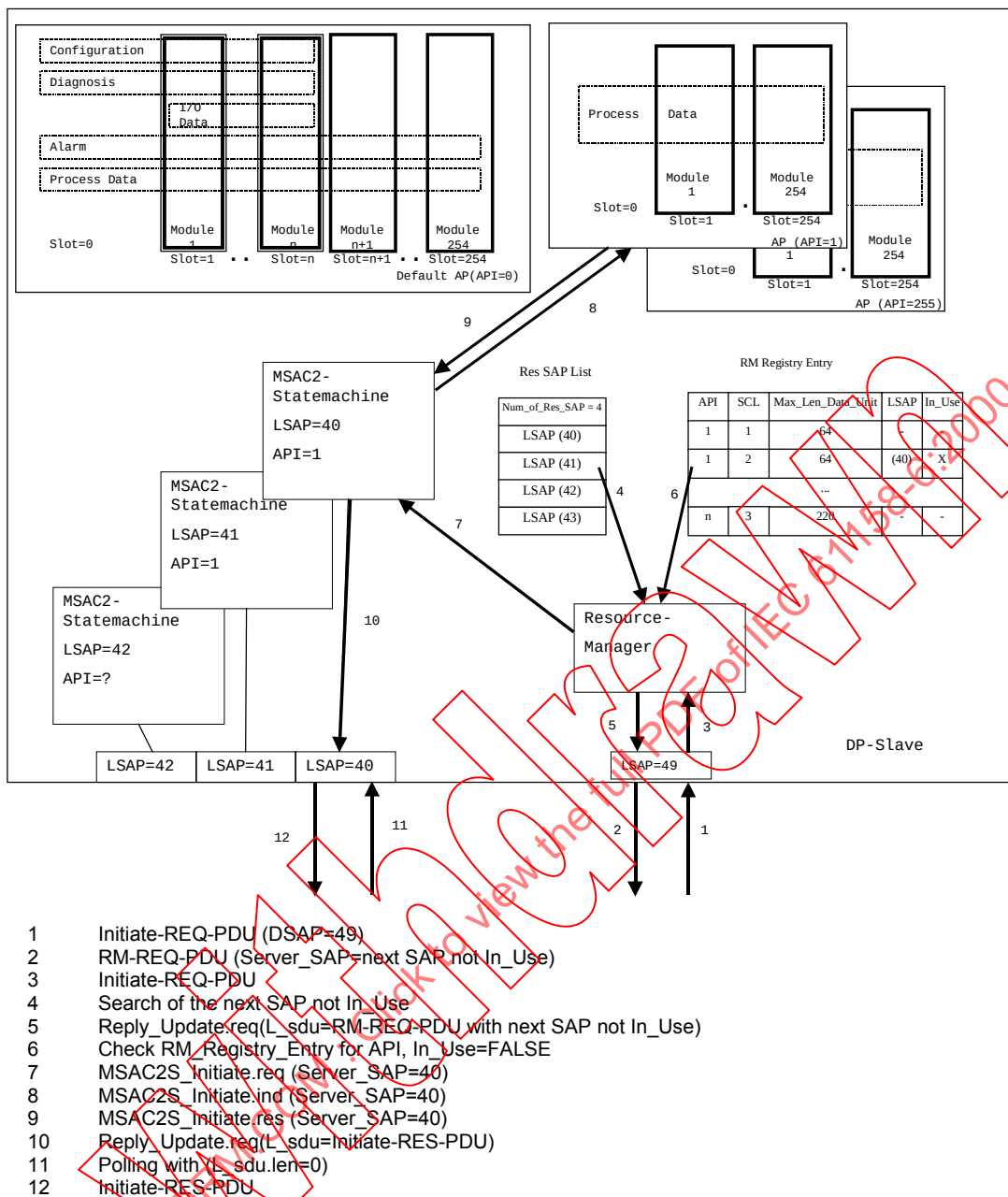


Figure 131 - Example for connection establishment on MS2(Server-Side)

The Resource Manager is responsible for the allocation of SAPs to incoming requests for MS2 AR communication.

If a Res_SAP is available when a Initiate-REQ-PDU is received on the RM_SAP, the Slave responds with a RM-REQ-PDU.

The only PDU which is accepted was an Initiate-REQ-PDU. The Resource Manager checks for an unused API/SCL in the RM_Registry and passes the Initiate-PDU with a Initiate to a MSAC2S State Machine.

If none of the entries in the RM_Registry fits to the requested API/SCL, the last SAP sent in the RM-REQ-PDU is temporarily used to send an Abort-REQ-PDU to the Master.

If no Res_SAP is available then RM_SAP is deactivated.

If an MSAC2S State Machine indicates with Abort.req that a connection has been closed, the corresponding entry in the RM_Registry is marked as unused and the corresponding DLSAP is released and can be used for further requests..

The following structures are used for

List of RM entries

static part		dynamic part	
API	SCL	Max_Len_Data_Unit	SAP
			In_Use

List of SSAPs

SSAP_Entry

(Unsigned8)

0..48

The State Machine needs some local variables which are described in clause a). Furthermore the machine uses functions and macros which are listed in clause b).

a) Local variables

Stored_Req_Add

Stored_Send_Timeout

Stored_Features_Supported

Stored_Profile_Features_Supported

Stored_Profile_Ident_Number

Stored_Add_Addr_Param

Storage of values of the last Initiate function (see part DP-Services).

RM_SAP

(constant)

Set to 49

Last_SAP

(Unsigned8)

0..48

SAP for previous established connection.

Server_SAP

(Unsigned8)

0..48

SAP which will be allocated to the next connection.

b) Macros

INITIATE-REQ-PDU-LEN

```
(
  48 {64 for linking device}
)
```

RMREQ-PDU-LEN

```
(
  4
)
```

6.6.3.3 MSRM2S State Table

Table 397 - MSRM2S State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MSRM2S_SInit.req =>RM_SAP:=49 Res_SAP_entry_Index:=1 Server_SAP:=Get_from_List_of_SSAPs() DMPMS_RSAP_ACTIVATE.req(SSAP=RM_SAP, Access=All, L_sdu_length_list=(<INITIATE-REQ-PDU-LEN>,0,<RMREQ-PDU-LEN>,0) Indication_Mode=Data)	WAIT-ACT
2	WAIT-ACT	DMPMS_RSAP_ACTIVATE.cnf(SSAP,M_status) /M_status=OK =>RM_SAP:=49 Res_SAP_entry_Index:=1 Server_SAP:=Get_from_List_of_SSAPs() DMPMS_REPLY_UPDATE.req(SSAP=RM_SAP, L_sdu=RM-REQ-PDU, Serv_class=Low, Transmit=Single)	INIT-WUPD
3	WAIT-ACT	DMPMS_RSAP_ACTIVATE.cnf(SSAP,M_status) /M_status=NO/IV =>MSRM2S_Fault.ind	POWER-ON
4	INIT-WUPD	DMPMS_REPLY_UPDATE.cnf (SSAP=RM_SAP, Serv_class=Low, L_status) /L_status=OK =>MSRM2S_SInit.cnf	OPEN
5	INIT-WUPD	DMPMS_REPLY_UPDATE.cnf (SSAP=RM_SAP, Serv_class=Low, L_status) /L_status=LS/IV/LR =>MSRM2S_Fault.ind	POWER-ON
6	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=RM_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu=Initiate-REQ-PDU && List_of_SSAPs<>empty =>Last_SAP := Server_SAP Stored_Req_Add:=Loc_add, Stored_Send_Timeout:= Send_Timeout Stored_Features_Supported:=Features_Supported Stored_Profile_Features_Supported:= Profile_Features_Supported Stored_Profile_Ident_Number:=Profile_Ident_Number Stored_Add_Addr_Param:= Add_Addr_Param Server_SAP:=Get_from_List_of_SSAPs() DMPMS_REPLY_UPDATE.req(SSAP=RM_SAP, L_sdu=RM-REQ-PDU, Serv_class=Low, Transmit=Single)	OW-UPDATE
7	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=RM_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu=Initiate-REQ-PDU && List_of_SSAPs=empty =>Stored_Req_Add:=Loc_add, Stored_Send_Timeout:= Send_Timeout Stored_Features_Supported:=Features_Supported Stored_Profile_Features_Supported:= Profile_Features_Supported Stored_Profile_Ident_Number:=Profile_Ident_Number Stored_Add_Addr_Param:= Add_Addr_Param DMPMS_SAP_DEACTIVATE.req (SSAP=RM_SAP)	OW-DEACT
8	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=RM_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && (L_sdu<>Initiate-REQ-PDU) =>ignore Lsdu DMPMS_REPLY_UPDATE.req(SSAP=RM_SAP, L_sdu=RM-REQ-PDU, Serv_class=Low, Transmit=Single)	INIT-WUPD
9	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=RM_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=NO =>ignore Lsdu	OPEN
10	OPEN	MSRM2S_Reset.req =>for (all CRL entries with AR-Type=MS2) MSAC2S_Reset.req(Res_SAP=SSAP) endfor	RESET-AC2
11	OPEN	MSAC2S_Closed.ind(Res_SAP) =>Index := Search_in_List_of_RM_Entries(SAP:=Res_SAP, In_Use:=True)	AW-SEARCH
12	AW-SEARCH	/Index <> 0 =>List_of_RM_Entries.In_Use[Index]:=False Put_to_List_of_SSAPs(Stored_Res_SAP)	OPEN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
13	AW-SEARCH	/Index = 0 =>MSRM2S_Fault.ind	POWER-ON
14	OW-UPDATE	DMPMS_REPLY_UPDATE.cnf (SSAP=RM_SAP, Serv_class=Low, L_status) /L_status=OK =>Index := Search_in_List_of_RM_Entries(API:=Add_Addr_Param.D_Addr.API, SCL:= Add_Addr_Param.D_Addr.SCL, In_Use:=False)	OW-SEARCH
15	OW-UPDATE	DMPMS_REPLY_UPDATE.cnf (SSAP=RM_SAP, Serv_class=Low, L_status) /L_status= LS/IV/LR =>MSRM2S_Fault.ind	POWER-ON
16	OW-SEARCH	/Index <> 0 =>List_of_RM_Entries.In_Use[Index]:=True List_of_RM_Entries.SAP[Index]:=Last_SAP Max_Len_Data_Unit:=List_of_RM_Entries.Max_Len_Data_Unit[Index] Send_Timeout:=Max(Stored_Send_Timeout, Stored_Min_Send_Timeout) MSAC2S_Initiate.req(Res_SAP=Last_SAP, Req_Add=Stored_Req_Add, Max_Len_Data_Unit, Send_Timeout, Features_Supported= Stored_Features_Supported, Profile_Features_Supported= Stored_Profile_Features_Supported, Profile_Ident_Number= Stored_Profile_Ident_Number, Add_Addr_Param= Stored_Add_Addr_Param)	OPEN
17	OW-SEARCH	/Index= 0 =>Send_Timeout:=Max(Stored_Send_Timeout, Stored_Min_Send_Timeout) MSAC2S_Abort.req(Res_SAP=Last_SAP, Req_Add, Subnet=False, Instance=MSAC2, Reason_Code=ABT_IA, Send_Timeout)	OPEN
18	OW-DEACT	DMPMS_SAP_DEACTIVATE.cnf(SSAP,M_status) /M_status=OK =>Index := Search_in_List_of_RM_Entries(API:=Add_Addr_Param.D_Addr.API, SCL:= Add_Addr_Param.D_Addr.SCL, In_Use:=False)	CW-SEARCH
19	OW-DEACT	DMPMS_SAP_DEACTIVATE.cnf(SSAP,M_status) /M_status=NO/IV =>MSRM2S_Fault.ind	POWER-ON
20	CW-SEARCH	/Index <> 0 =>List_of_RM_Entries.In_Use[Index]:=True List_of_RM_Entries.SAP[Index]:=Last_SAP Max_Len_Data_Unit:=List_of_RM_Entries.Max_Len_Data_Unit[Index] Send_Timeout:=Max(Stored_Send_Timeout, Stored_Min_Send_Timeout) MSAC2S_Initiate.req(Res_SAP=Last_SAP, Req_Add=Stored_Req_Add, Max_Len_Data_Unit, Send_Timeout, Features_Supported= Stored_Features_Supported, Profile_Features_Supported= Stored_Profile_Features_Supported, Profile_Ident_Number= Stored_Profile_Ident_Number, Add_Addr_Param= Stored_Add_Addr_Param)	CLOSED
21	CW-SEARCH	/Index = 0 =>Send_Timeout:=Max(Stored_Send_Timeout, Stored_Min_Send_Timeout) MSAC2S_Abort.req(Res_SAP=Last_SAP, Req_Add, Subnet=False, Instance=MSAC2, Reason_Code=ABT_IA, Send_Timeout)	CLOSED
22	CLOSED	MSRM2S_Reset.req =>for (all CRL entries with AR-Type=MS2) MSAC2S_Reset.req(Res_SAP=SSAP) endfor	RESET-AC2
23	CLOSED	MSAC2S_Closed.ind(Res_SAP) =>Stored_Res_SAP:=Res_SAP Index := Search_in_List_of_RM_Entries(SAP:=Res_SAP, In_Use:=True)	REOW-SEARCH
24	REOW-SEARCH	/Index <> 0 =>List_of_RM_Entries.In_Use[Index]:=False Put_to_List_of_SSAPs(Stored_Res_SAP)	WAIT-ACT
25	REOW-SEARCH	/Index = 0 =>MSRM2S_Fault.ind	POWER-ON
26	RESET-AC2	for (all CRL entries with AR-Type=MS2) MSAC2S_Reset.cnf(Res_SAP=SSAP) endfor /M_status=OK =>MSRM2S_Reset.cnf	POWER-ON
27	ANY-STATE	MSAC2S_Fault.ind(Res_SAP) =>MSRM2S_Fault.ind	POWER-ON
28	ANY-STATE	unexpected DL reaction =>MSRM2S_Fault.ind	POWER-ON

6.6.4 MSAC2S

6.6.4.1 Primitive Definitions

6.6.4.1.1 Primitives Exchanged between MSAC2S and FSPMS

Table 398 - Primitives issued by FSPMS to MSAC2S

Primitive Name	Source	Associated Parameters	Functions
Initiate.rsp(+)	FSPMS	Res SAP, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	Refer to FAL Type 3 Service Definition IEC 61158-5
Initiate.rsp(-)	FSPMS	Res SAP, Error Decode, Error Code 1 Error Code 2	
Read.rsp(+)	FSPMS	Res SAP, Length, Data	
Read.rsp(-)	FSPMS	Res SAP, Error Decode, Error Code 1 Error Code 2	
Write.rsp(+)	FSPMS	Res SAP, Length	
Write.rsp(-)	FSPMS	Res SAP, Error Decode, Error Code 1 Error Code 2	
Data Transport.rsp(+)	FSPMS	Res SAP, Length, Data	
Data Transport.rsp(-)	FSPMS	Res SAP, Error Decode, Error Code 1 Error Code 2	
Abort.req	FSPMS	Res SAP, Subnet, Instance, Reason Code	

Table 399 - Primitives issued by MSAC2S to FSPMS

Primitive Name	Source	Associated Parameters	Functions
Initiate.ind	MSAC2S	Res SAP, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	Refer to FAL Type 3 Service Definition IEC 61158-5
Abort.ind	MSAC2S	Res SAP, Locally Generated, Subnet, Instance, Reason Code, Additional Detail	
Read.ind	MSAC2S	Res SAP, Slot Number, Index, Length	
Write.ind	MSAC2S	Res SAP, Slot Number, Index, Length, Data	
Data Transport.ind	MSAC2S	Res SAP, Slot Number, Index, Length, Data	

6.6.4.1.2 Primitives Exchanged between MSAC2S and MSRM2S**Table 400 - Primitives issued by MSRM2S to MSAC2S**

Primitive Name	Source	Associated Parameters	Functions
Reset.req	MSRM2S	Res SAP	Refer to FAL Type 3 Service Definition IEC 61158-5
Initiate.req	MSRM2S	Res SAP, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
RM2_Abort.req	MSRM2S	Res SAP, Subnet, Instance, Reason Code	

Table 401 - Primitives issued by MSAC2S to MSRM2S

Primitive Name	Source	Associated Parameters	Functions
Reset.cnf	MSAC2S	Res SAP	Refer to FAL Type 3 Service Definition IEC 61158-5
Fault.ind	MSAC2S	Res SAP	
Closed.ind	MSAC2S	Res SAP	Signals the Termination of a MS2 Communication

6.6.4.1.3 Parameters of MSAC2S Primitives

The parameters used with the primitives exchanged between the MSAC2S and FSPMS,MSRM2S are described in FAL Type 3 Service Definition IEC 61158-5

Table 402 – Parameter Used with Primitives Exchanged with MSAC2S

Parameter Name	Description
Res SAP	This parameter conveys the SSAP for addressing of the various MS2 AR.

6.6.4.2 State Machine Description

In the state CLOSED the Initiate service is expected from the Resource Manager. Then the corresponding DLSAP is activated and the User gets the Initiate.ind. The User shall generate a MSAC2S_Initiate.rsp and the connection is established after the Master has fetched the response.

In the OPEN-state valid services (see macro <VALID_SERVICE>) are allowed. Only one request from the DP-Master is processed simultaneously, otherwise the connection is aborted.

Both the DP-Master and the DP-Slave can close the connection using the Abort. The remote User gets an Abort.ind.

The connection monitoring is based on the U-Timer, F-Timer and I-Timer (see appendix "Monitoring of Communication Relationships")

The State Machine needs some local variables which are described in clause a). Furthermore the machine uses functions and macros which are listed in clause b).

a) Local variables of the MSAC2S**Server_SAP**

(Unsigned8)

The used SAP for the MS2 connection.

U-Timer

(Unsigned16)

The U-Timer controls the reaction of the User after delivering an indication. If the U-Timer expires an Idle-PDU is created and transferred to the local DLL. The U-Timer is stopped when the User provides the response.

F-Timer

(Unsigned16)

The F-Timer controls the Master fetching the previously provided PDU. If the F-Timer expires an Abort is generated, transferred to the local DLL and the F-Timer starts again. The F-Timer is stopped as soon as the Master fetches the PDU.

I-Timer

(Unsigned16)

The I-Timer controls the next activity of the client after the Master has fetched the previous PDU. When the I-Timer expires an Abort is generated, transferred to the local DLL and the F-Timer starts. The I-Timer is stopped when the Master has sent the next request.

NOTE Only one of the timers described above is running at the same time.

Service_Header

(Unsigned8)

The actual service header to check whether the User generates the correct response.

Return-State

INITIATE, VALID-SERVICE

This variable is used to store the state where to return after an idle cycle has been processed.

Go_Abort

(Boolean)

Local flag that indicates an abort reaction after finishing the present service response.

Service_Buffer

(Octet-String[244])

Local buffer that stores one complete Response-PDU (Initiate.rsp or <VALID_SERVICE>.rsp) during an outstanding Idle-RES-PDU.

Stored_Slot_Number

Stored_Index

Stored values of the parameters of Read/Write-services

Stored_Instance

Stored_Reason

Stored values of the parameters of MSAC2S_Abort-service

INITIATE-REQ-PDU-LEN

(local constant)

PDU-size (the 4 Byte Header is excluded) set to 48 (64 respectively for links)

ABORT_PDU_LEN

(local constant)

Length of an Abort-REQ-PDU = 4

b) Macros**Read-REQ-PDU**

This function will return True if the corresponding L_sdu is a valid Read-REQ-PDU

Write-REQ-PDU

This function will return True if the corresponding L_sdu is a valid Write-REQ-PDU

Data_Transport-REQ-PDU

This function will return True if the corresponding L_sdu is a valid Data_Transport-REQ-PDU

Abort-REQ-PDU

This function will return True if the corresponding L_sdu is a valid Abort-REQ-PDU

Idle-REQ-PDU

This function will return True if the corresponding L_sdu is a valid Idle-REQ-PDU

STORE_ABORT_PARAMETER

```
(
  Stored-Instance=Instance
  Stored-Reason_Code=Reason_Code
)
```

<VALID-SERVICE>

```
<
  Service_Header=0x5E: Read
  Service_Header=0x5F: Write
  Service_Header=0x51: Data_Transport
>
```

VALID-SERVICE-REQ-PDU

```
(
  (Read-REQ-PDU OR Write-REQ-PDU OR Data_Transport-REQ-PDU)
  && Stored_Max_Len_Data_Unit ≥ Length
)
```


6.6.4.3 MSAC2S State Table

Table 403 - MSAC2S State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	=>Server_SAP:=Res_SAP	CLOSED
2	CLOSED	MSAC2S_Initiate.req(Res_SAP, Req_Add, Max_Len_Data_Unit, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>Service_Header:=0x57 F-Timer:=Send_Timeout U-Timer:=Send_Timeout I-Timer:=2*Send_Timeout Stored_Max_Len_Data_Unit:= Max(Max_Len_Data_Unit, INITIATE-REQ-PDU-LEN) clear Service_Buffer Go_Abort:=FALSE MSAC2_Initiate.ind(Req_Add, Res_SAP, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) DMPMS_RSAP_ACTIVATE.req(SSAP=Server_SAP, Access=Req_Add, L_sdu_length_list=(Max_Len_Data_Unit,0, Max_Len_Data_Unit,0), Indication_Mode=Data)	SET-ACC- POS
3	CLOSED	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) =>ignore	CLOSED
4	CLOSED	msac2s_Abort.req(Res_SAP, Req_Add, Subnet, Instance, Reason_Code, Send_Timeout) =>F-Timer:=Send_Timeout STORE_ABORT_PARAMETER DMPMS_RSAP_ACTIVATE.req(SSAP=Server_SAP, Access=Req_add, L_sdu_length_list=(ABORT_PDU_LEN,0, ABORT_PDU_LEN,0) Indication_Mode=Data)	SET-ACC-FE
5	CLOSED	MSAC2S_XXX.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	CLOSED
6	CLOSED	MSAC2S_Initiate.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	CLOSED
7	CLOSED	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>ignore	CLOSED
8	SET-ACC- POS	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status = OK Start U-Timer	INITI-WRES
9	SET-ACC- POS	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status=NO/IV =>MSAC2S_Fault.ind	POWER-ON
10	SET-ACC-FE	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status = OK =>L_sdu:=Abort-REQ-PDU(with stored parameter) Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT- WUPD
11	SET-ACC-FE	DMPMS_RSAP_ACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status=NO/IV =>MSAC2S_Fault.ind	POWER-ON
12	INITI-WRES	MSAC2S_Initiate.rsp(+)(Res_SAP) =>L_sdu:=Initiate-RES-PDU(With Stored_Max_Len_Data_Unit) Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	INITI-WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
13	INITI-WRES	MSAC2S_Initiate.rsp(-)(Res_SAP) =>L_sdu:=Initiate-NRS-PDU Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
14	INITI-WRES	MSAC2S_XXX.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD
15	INITI-WRES	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>L_sdu:=Abort-REQ-PDU Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
16	INITI-WRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add, L_sdu, Serv_class=Low, Update_status) /Abort-REQ-PDU =>Stop U-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU)	WAIT-DEACT
17	INITI-WRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add, L_sdu, Serv_class=Low, Update_status) /NOT(Abort-REQ-PDU) =>Reason_Code:=ABT_FE Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD
18	INITI-WRES	U-Timer expired =>L_sdu:=Idle-REQ-PDU Return_State:=INITIATE Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	IDLE-WUPD
19	INITI-WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /L_status=OK =>ignore	INITI-SRES
20	INITI-WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /L_status=LS/IV/LR =>Stop F-Timer Stop I-Timer MSAC2S_Fault.ind	POWER-ON
21	INITI-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add, L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 && Go_Abort = FALSE =>Stop F-Timer Start I-Timer	OPEN
22	INITI-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add, L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 && Go_Abort = TRUE =>L_sdu:=Abort-REQ-PDU(With Stored Parameter) Stop F-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
23	INITI-SRES	DMPMS_DATA_REPLY.ind(SSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO/NO && L_sdu.len<>0 && Abort-REQ-PDU =>Stop F-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU)	WAIT-DEACT
24	INITI-SRES	DMPMS_DATA_REPLY.ind(SSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len<>0 && NOT(Abort-REQ-PDU) =>Reason_Code:=ABT_FE Stop F-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD
25	INITI-SRES	DMPMS_DATA_REPLY.ind(SSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=NO && L_sdu.len<>0 && NOT(Abort-REQ-PDU) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_FE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	INITI-SRES
26	INITI-SRES	MSAC2S_XXX.rsp(+/-)(Res_SAP) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	INITI-SRES
27	INITI-SRES	MSAC2S_Initiate.rsp(+/-)(Res_SAP) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	INITI-SRES
28	INITI-SRES	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER	INITI-SRES
29	INITI-SRES	F-Timer expired =>Reason_Code:=ABT_TO Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD
30	IDLE-WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /Status=OK	IDLE-SREQ
31	IDLE-WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /L_status= LS/IV/LR =>Stop F-Timer Stop I-Timer MSAC2S_Fault.ind	POWER-ON
32	IDLE-SREQ	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 && Go_Abort = FALSE =>Stop F-Timer Start I-Timer	W-IDLE-CON
33	IDLE-SREQ	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 && Go_Abort = TRUE =>L_sdu:=Abort-REQ-PDU(With Stored Parameter) Stop F-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT- WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
34	IDLE-SREQ	DMPMS_DATA_REPLY.ind(SSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO/NO && L_sdu.len<>0 && Abort-REQ-PDU ddlms_msac2s_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2_Abort.ind(Res_SAP=Server_SAP, Locally-Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU) Stop F-Timer	WAIT-DEACT
35	IDLE-SREQ	DMPMS_DATA_REPLY.ind(SSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len<>0 && NOT(Abort-REQ-PDU) =>Reason_Code:=ABT_FE Stop F-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD
36	IDLE-SREQ	DMPMS_DATA_REPLY.ind(SSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=NO && L_sdu.len<>0 && NOT(Abort-REQ-PDU) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_FE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	IDLE-SREQ
37	IDLE-SREQ	MSAC2S_XXX.rsp(+)(Res_SAP) /Service-Response fits stored Service_Header && Service_Buffer empty =>Service_Buffer:= <VALID-SERVICE>-RES-PDU(with Stored_Slot_Number,Stored_Index)	IDLE-SREQ
38	IDLE-SREQ	MSAC2S_XXX.rsp(-)(Res_SAP) /Service-Response fits stored Service_Header && Service_Buffer empty =>Service_Buffer:= <VALID-SERVICE>-NRS-PDU(with Stored_Slot_Number,Stored_Index)	IDLE-SREQ
39	IDLE-SREQ	MSAC2S_Initiate.rsp(+)(Res_SAP) /Service-Response fits stored Service_Header && Service_Buffer empty =>Service_Buffer:= Initiate-RES-PDU	IDLE-SREQ
40	IDLE-SREQ	MSAC2S_Initiate.rsp(-)(Res_SAP) /Service-Response fits stored Service_Header && Service_Buffer empty =>Service_Buffer:= Initiate-NRS-PDU	IDLE-SREQ
41	IDLE-SREQ	MSAC2S_XXX.rsp(+/-)(Res_SAP) /(NOT(Service-Response fits stored Service_Header) NOT(Service_Buffer empty)) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_RE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	IDLE-SREQ
42	IDLE-SREQ	MSAC2S_Initiate.rsp(+/-)(Res_SAP) /(NOT(Service-Response fits stored Service_Header) NOT(Service_Buffer empty)) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_RE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	IDLE-SREQ
43	IDLE-SREQ	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER	IDLE-SREQ
44	IDLE-SREQ	F-Timer expired =>Reason_Code:=ABT_TO Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
45	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-RES-PDU && Return-State=INITIATE && Go_Abort=FALSE && (Service_Buffer empty) =>Stop I-Timer Start U-Timer	INITI-WRES
46	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-RES-PDU && Return-State=INITIATE && Go_Abort=FALSE && NOT(Service_Buffer empty) && Service_Buffer=Initiate-NRS-PDU =>L_sdu:=Stored L_sdu from Service_Buffer clear Service_Buffer Stop I-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
47	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-RES-PDU && Return-State=INITIATE && Go_Abort=FALSE && NOT(Service_Buffer empty) && Service_Buffer=Initiate-RES-PDU =>L_sdu:=Stored L_sdu from Service_Buffer clear Service_Buffer Stop I-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	INITI-WUPD
48	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-RES-PDU && Return-State=VALID-SERVICE && Go_Abort=FALSE && (Service_Buffer empty) =>Stop I-Timer Start U-Timer	VS-WRES
49	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-RES-PDU && Return-State=VALID-SERVICE && Go_Abort=FALSE && NOT(Service_Buffer empty) =>L_sdu:=Stored L_sdu from Service_Buffer clear Service_Buffer Stop I-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	VS-WUPD
50	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-RES-PDU && Go_Abort=TRUE =>L_sdu:=Abort-REQ-PDU(with stored Parameter) DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
51	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Abort-REQ-PDU =>Stop I-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU)	WAIT-DEACT
52	W-IDLE-CON	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /NOT(Abort-REQ-PDU Idle-RES-PDU) =>Reason_Code:=ABT_FE Stop I-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
53	W-IDLE-CON	MSAC2S_XXX.rsp(+)(Res_SAP) /Service-Response fits stored Service_Header && Service_Buffer empty =>Service_Buffer:= <VALID-SERVICE>-RES-PDU(with Stored_Slot_Number,Stored_Index)	W-IDLE-CON
54	W-IDLE-CON	MSAC2S_XXX.rsp(-)(Res_SAP) /Service-Response fits stored Service_Header && Service_Buffer empty =>Service_Buffer:= <VALID-SERVICE>-NRS-PDU (with Stored_Slot_Number,Stored_Index)	W-IDLE-CON
55	W-IDLE-CON	MSAC2S_XXX.rsp(+/-)(Res_SAP) /(NOT(Service-Response fits stored Service_Header) NOT(Service_Buffer empty)) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_RE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	W-IDLE-CON
56	W-IDLE-CON	MSAC2S_Initiate.rsp(+/-)(Res_SAP) /(NOT(Service-Response fits stored Service_Header) NOT(Service_Buffer empty)) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_RE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	W-IDLE-CON
57	W-IDLE-CON	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER	W-IDLE-CON
58	W-IDLE-CON	I-Timer expired =>Reason_Code:=ABT_TO Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD
59	SABORT- WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /Status=OK	SABORT- SRES
60	SABORT- WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /L_status=LS/IV/LR =>Stop F-Timer Stop I-Timer MSAC2S_Fault.ind	POWER-ON
61	SABORT- SRES	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>ignore	SABORT- SRES
62	SABORT- SRES	MSAC2S_XXX.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- SRES
63	SABORT- SRES	MSAC2S_Initiate.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- SRES
64	SABORT- SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 =>Stop I-Timer Stop F-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	WAIT-DEACT
65	SABORT- SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=NO && L_sdu.len<>0 =>ignore	SABORT- SRES

#	Current State	Event /Condition =>Actions taken&&Output	Next State
66	SABORT-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len<>0 =>ignore L_sdu Stop I-Timer Stop F-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	WAIT-DEACT
67	SABORT-SRES	I-Timer expired =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code=ABT_TO)	WAIT-DEACT
68	SABORT-SRES	F-Timer expired =>DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code=ABT_TO)	WAIT-DEACT
69	WAIT-DEACT	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status = OK =>MSAC2S_Closed.ind(Res_SAP)	CLOSED
70	WAIT-DEACT	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP,M_status) /M_status=NO/IV =>MSAC2S_Fault.ind	POWER-ON
71	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /VALID-SERVICE-REQ-PDU =>Service_Header:=Function_Num Stored_Slot_Number:=Slot_Number Stored_Index:=Index Stop I-Timer Start U-Timer MSAC2S_<VALID-SERVICE>.ind(Res_SAP=Server_SAP)	VS-WRES
72	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Idle-REQ-PDU {IDLE-PDU} =>L_sdu:=Idle-RES-PDU Stop I-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	VS-WUPD
73	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Abort-REQ-PDU =>Stop I-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU)	WAIT-DEACT
74	OPEN	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /NOT(VALID-SERVICE-REQ-PDU (Abort-REQ-PDU) (Idle-REQ-PDU)) =>Reason_Code:=ABT_FE Stop I-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD
75	OPEN	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>L_sdu:=Abort-REQ-PDU Stop I-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
76	OPEN	MSAC2S_Initiate.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE Stop I-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
77	OPEN	MSAC2S_XXX.rsp(+/-)(Res_SAP) =>L_sdu:=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code=ABT_TO) Stop I-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
78	OPEN	I-Timer expired =>Reason_Code:=ABT_TO Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD
79	VS-WRES	MSAC2S_XXX.rsp(+)(Res_SAP) /Service-Response fits stored Service_Header && Stored_Max_Len_Data_Unit □ Length =>L_sdu:=<VALID-SERVICE>-RES-PDU(with Stored_Slot_Number,Stored_Index) Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	VS-WUPD
80	VS-WRES	MSAC2S_XXX.rsp(-)(Res_SAP) /Service-Response fits stored Service_Header && Stored_Max_Len_Data_Unit □ Length =>L_sdu:=<VALID-SERVICE>-NRS-PDU Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	VS-WUPD
81	VS-WRES	MSAC2S_Initiate.rsp(+/-)(Res_SAP) =>Reason_Code:=ABT_SE Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD
82	VS-WRES	MSAC2S_XXX.rsp(+/-)(Res_SAP) /NOT(Service Response fits stored Service_Header && Stored_Max_Len_Data_Unit □ Length) =>Reason_Code:=ABT_RE Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD
83	VS-WRES	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>L_sdu:=Abort-REQ-PDU Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT-WUPD
84	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Abort-REQ-PDU =>Stop U-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU)	WAIT-DEACT
85	VS-WRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /NOT(Abort-REQ-PDU) =>Reason_Code:=ABT_FE Stop U-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT-WUPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
86	VS-WRES	U-Timer expired =>L_sdu:=Idle-REQ-PDU Return-State:=VALID-SERVICE Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	IDLE-WUPD
87	VS-WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /Status=OK	VS-SRES
88	VS-WUPD	DMPMS_REPLY_UPDATE.cnf(SSAP=Server_SAP, Serv_class=Low, L_status) /L_status= LS/IV/LR =>Stop F-Timer Stop I-Timer MSAC2S_Fault.ind	POWER-ON
89	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 && Go_Abort = FALSE Stop F-Timer Start I-Timer	OPEN
90	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len=0 && Go_Abort = TRUE =>L_sdu:=Abort-REQ-PDU with stored Parameter Stop F-Timer Start I-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu, Serv_class=Low, Transmit=Single)	SABORT- WUPD
91	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO/NO && L_sdu.len<>0 && Abort-REQ-PDU =>Stop F-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP) MSAC2S_Abort.ind(Res_SAP=Server_SAP Locally_Generated=FALSE, (Subnet, Instance, Reason_Code) from Abort-REQ-PDU)	WAIT-DEACT
92	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=LO && L_sdu.len<>0 && NOT (Abort-REQ-PDU) =>Reason_Code:=ABT_FE Stop F-Timer Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD
93	VS-SRES	DMPMS_DATA_REPLY.ind(SSAP, DSAP=Server_SAP, Loc_add, Rem_add,L_sdu, Serv_class=Low, Update_status) /Update_status=NO && L_sdu.len<>0 && NOT(Abort-REQ-PDU) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_FE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	VS-SRES
94	VS-SRES	MSAC2S_XXX.rsp(+/-)(Res_SAP) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	VS-SRES
95	VS-SRES	MSAC2S_Initiate.rsp(+/-)(Res_SAP) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER Reason_Code:=ABT_SE MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	VS-SRES
96	VS-SRES	MSAC2S_Abort.req(Res_SAP, Subnet, Instance, Reason_Code) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER	VS-SRES

#	Current State	Event /Condition =>Actions taken&&Output	Next State
97	VS-SRES	F-Timer expired =>Reason_Code:=ABT_TO Start F-Timer DMPMS_REPLY_UPDATE.req(SSAP=Server_SAP, L_sdu=Abort-REQ-PDU(Subnet=NO, Instance=MSAC2, Reason_Code), Serv_class=Low, Transmit=Single) MSAC2S_Abort.ind(Res_SAP=Server_SAP, Locally_Generated=TRUE, Subnet=NO, Instance=MSAC2, Reason_Code)	SABORT- WUPD
98	ANY-STATE	unexpected DMPMS reaction =>Stop U-Timer Stop F-Timer Stop I-Timer MSAC2S_Fault.ind	POWER-ON
99	ANY-STATE	msac2s_Go_Power_On.req(Res_SAP, Reason) /Reason = FDL_Fault =>Stop U-Timer Stop F-Timer Stop I-Timer	POWER-ON
100	ANY-STATE	msac2s_Go_Power_On.req(Res_SAP, Reason) /Reason <> FDL_Fault =>Stop U-Timer Stop F-Timer Stop I-Timer DMPMS_SAP_DEACTIVATE.req(SSAP=Server_SAP)	DEACT-PON
101	DEACT-PON	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP, M_status) /M_status = OK	POWER-ON
102	DEACT-PON	DMPMS_SAP_DEACTIVATE.cnf(SSAP=Server_SAP, M_status) /M_status=NO/IV =>MSAC2S_Fault.ind	POWER-ON

6.6.5 MSCY1M

6.6.5.1 Primitive Definitions

6.6.5.1.1 Primitives Exchanged between FSPMM1 and MSCY1M

Table 404 - Primitives issued by FSPMM1 and MSCY1M

Primitive Name	Source	Associated Parameters	Functions
MInit MS0.req	FSPMM1	Rem Add	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMM1	(none)	
Abort.req	FSPMM1	Rem Add, Subnet, Instance, Reason Code	
Start Slave Handler.req	FSPMM1	Rem Add	Start first Cycle of MSCY1M
Stop Slave Handler.req	FSPMM1	Rem Add	Stop processing of MSCY1M
Cont Slave Handler.req	FSPMM1	Rem Add, Output Clear	Start new Cycle of MSCY1M
Get Slave Diag.req	FSPMM1	Rem Add	Refer to FAL Type 3 Service Definition IEC 61158-5
Set Output.req	FSPMM1	Rem Add, Slot_Number Output Data	
Get Input.req	FSPMM1	Rem Add, Slot Number	

Table 405 - Primitives issued by MSCY1M to FSPMM1

Primitive Name	Source	Associated Parameters	Functions
MInit MS0.cnf	MSCY1M	Rem Add	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MSCY1M	(none)	
Start Slave Handler.cnf	MSCY1M	Rem Add	Start first Cycle of MSCY1M
Stop Slave Handler.cnf	MSCY1M	Rem Add	Stop processing of MSCY1M
Cont Slave Handler.cnf	MSCY1M	Rem Add, Diag, No ACIrr	Start new Cycle of MSCY1M
Get Slave Diag.cnf(+)	MSCY1M	Rem Add, CREP, Diag Data	Refer to FAL Type 3 Service Definition IEC 61158-5
Get Slave Diag.cnf(-)	MSCY1M	Rem Add	
Set Output.cnf(+)	MSCY1M	Rem Add, Slot Number	
Set Output.cnf(-)	MSCY1M	Rem Add, Slot Number	
Get Input.cnf(+)	MSCY1M	Rem Add, Slot_Number, Input Data	
Get Input.cnf(-)	MSCY1M	Rem Add	
New Slave Diag.ind	MSCY1M	Rem Add	
New Input.ind	MSCY1M	Rem Add	
Started.ind	MSCY1M	Rem Add, Alarm Limit	
Stopped.ind	MSCY1M	Rem Add	
Fault.ind	MSCY1M	Rem Add	

6.6.5.1.2 Parameters of MSCY1M Primitives

Table 406 – Parameters Used with Primitives Exchanged between FSPMM1 and MSCY1M

Parameter Name	Description
Rem Add	This parameter is used to identify the DP-Slave which is assigned to that communication relation.
Output Clear	This parameter indicates MSCY1M that the Outputs shall be cleared.
Diag	This parameter indicates FSPMM1 that a no Data Exchange was processed in the preceeding cycle or the Slave was deactivated.
No ACIrr	This parameter indicates FSPMM1 that the corresponding DP-Slave is operated in the non Autoclear-Mode (this DP-Slave has no influence to the Masters State (operate, clear).

Other parameters used with the primitives exchanged between the FSPMM1 and the MSCY1M are described in FAL Type 3 Service Definition IEC 61158-5.

6.6.5.2 State Machine Description

For each possible DP-Slave a State Machine MSCY1M (Slave-Handler) is established and started by a superior State Machine, the FSPMM1. The remote address of the Slave (Rem_Add) is used as a local reference for MSCY1M.

The Slave-Handler manages the individual states for every DP-Slave. The following main states will be distinguished:

- diagnostic
- parameterisation
- configuration
- data exchange

SI_Para_Exist

Access: -r

Local information indicating whether or not a Slave parameter set of the associated Rem_Add exists.

SPara

Access: -r/-w

Any Information from ARL/CRL related to this DP-Slave.

BOutput

Access: -r

Local storage of Output data for the DP-Slave.

BInput

Access: -w

Local storage of Input data for the DP-Slave.

BDiag

Access: -r/-w

Local storage of Diagnostic information of the DP-Slave.

MSCY1M changes the following flags of the diagnostic information:

Invalid_Slave_Response

Station_Non_Existent

Deactivated

DTrans

This variable is mapped on the Data Transfer List attribute of CR-List.

Access: -r/-w

The user data exchange mode between the DP-Master (Class 1) and its assigned DP-Slaves is supervised on the Master side. This means it is checked whether an user data transfer to the associated DP-Slaves was executed within the last DP cycle or a diagnostic cycle, after which the directly following Slave state is again the data transfer state. Any other following state forces the associated bit of the Slave to be cleared.

SDiag

This variable is mapped on the System Diagnosis List attribute of CR-List.

Access: -r/-w

It is set while the MSCY1M state machine of an activated Slave is not able to establish the connection to the Slave without errors. E. g. it is set if a Slave does not respond, if he responds with a negative DL confirmation or if he does not respond with the correct number of input data during Data_Exchange. The bit is reset after the connection could be established without any error. It is also reset if the Slave is deactivated by the User or if the MSCY1M State Machine is stopped

Local variables**Delay_Count**

(Unsigned8)

The Delay_Count is used to count the number of Slave_Diag.cnf in the state DIAG2 while Diag_Data.Prm_Req is still set (Slow Slave). The Delay_Count is initialised with the value of SPara.Diag_Upd_Delay.

Range: 0 to 15 (extendible up to 255)

MSAL1M_Started

(Boolean)

This variable is used to store the information whether the MSAL1M State Machine is started (True) or not (False).

Wait_for_Start_con

(Boolean)

This variable is used to store the information whether a Start.req has not yet been confirmed by the MSAL1M State Machine (True) or not (False).

Go-Abort

(Boolean)

This variable is used to store the information that an error occurred forcing the MSCY1M State Machine to abort the connection.

Macros

CHANGE_DIAG_IND

```
(
  MSCY1M_New Slave Diag.ind(Rem_Add)
  IF (MSAL1M_Started = TRUE) THEN
    (
      MSAL1M_Change Diag.req(Rem_Add)
    )
  )
```

GO_ABORT

```
(
  DTrans = 0
  Go_Abort = TRUE
  Blnp = Nil
)
```

START_MSAL1M

```
(
  IF (SPara.DPV1_Supported= TRUE) THEN
    (
      IF (Wait_for_Start_con = FALSE) THEN
        (
          IF (MSAL1M_Started = FALSE) THEN
            (
              MSAL1M_Start.req(Rem_Add)
              MSAL1M_Started = TRUE
            )
          )
        )
      )
    )
  )
```

6.6.5.3 MSCY1M State Table

Table 407 - MSCY1M State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MSCY1M Minit MS0.req (Rem Add) =>BDiag := NIL, SDiag := 0, DTrans := 0 Go_Abort := FALSE MSAL1M_Started := FALSE MSCY1M Minit MS0.cnf(Rem Add)	STOP
2	STOP	MSCY1M Abort.req (Rem Add) =>ignore	STOP
3	STOP	MSAL1 Abort.ind (Rem_Add) =>ignore	STOP
4	STOP	MSCY1M Start Slave Handler.req (Rem Add) /SI_Para_Exist = FALSE SPara.Active = FALSE =>MSCY1M Start Slave Handler.cnf(Rem Add)	DEACT
5	STOP	MSCY1M Start Slave Handler.req (Rem Add) /SI_Para_Exist = TRUE && SPara.Active = TRUE =>SDiag := 1, BInput := Nil, BOutput := Nil BDiag.Deactivated := FALSE BDiag.Station_Non_Exist := TRUE MSCY1M Start Slave Handler.cnf(Rem Add)	DIAG1
6	STOP	MSCY1M Get Slave Diag.req (Rem Add) =>MSCY1M Get Slave Diag.cnf(-) (Rem Add)	STOP
7	STOP	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>MSCY1M Set Output.cnf(-) (Rem Add, Slot Number)	STOP
8	STOP	MSCY1M Get Input.req (Rem Add, Slot Number) =>MSCY1M Get Input.cnf(-) (Rem Add)	STOP
9	DEACT	MSCY1M Abort.req (Rem Add) =>ignore	DEACT
10	DEACT	MSAL1 Abort.ind (Rem_Add) =>ignore	DEACT
11	DEACT	MSCY1M Stop Slave Handler.req (Rem Add) =>MSCY1M Stop Slave Handler.cnf(Rem Add)	STOP
12	DEACT	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /SI_Para_Exist = FALSE SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DEACT
13	DEACT	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /SI_Para_Exist = TRUE && SPara.Active = TRUE =>SDiag := 1, BInput := Nil, BOutput := Nil BDiag.Deactivated := FALSE BDiag.Station_Non_Exist := TRUE DMPMM1 Slave Diag.req (Rem Add)	WDIAG1
14	DEACT	MSCY1M Get Slave Diag.req (Rem Add) =>MSCY1M Get Slave Diag.cnf(-) (Rem Add)	DEACT
15	DEACT	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>MSCY1M Set Output.cnf(-) (Rem Add, Slot Number)	DEACT
16	DEACT	MSCY1M Get Input.req (Rem Add, Slot Number) =>MSCY1M Get Input.cnf(-) (Rem Add)	DEACT
17	DIAG1	MSCY1M Abort.req (Rem Add) =>ignore	DIAG1
18	DIAG1	MSAL1 Abort.ind (Rem_Add) =>ignore	DIAG1
19	DIAG1	MSCY1M Stop Slave Handler.req (Rem Add) =>BDiag.Deactivated := TRUE, SDiag := 0 MSCY1M Stop Slave Handler.cnf(Rem Add)	STOP
20	DIAG1	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) =>DMPMM1 Slave Diag.req (Rem Add)	WDIAG1
21	DIAG1	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	DIAG1

#	Current State	Event /Condition =>Actions taken&&Output	Next State
22	DIAG1	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	DIAG1
23	DIAG1	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	DIAG1
24	WDIAG1	MSCY1M Abort.req (Rem Add) =>ignore	WDIAG1
25	WDIAG1	MSAL1 Abort.ind (Rem_Add) =>ignore	WDIAG1
26	WDIAG1	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WDIAG1
27	WDIAG1	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WDIAG1
28	WDIAG1	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WDIAG1
29	WDIAG1	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE BDiag.Deactivated := TRUE, SDiag := 0 MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	DEACT
30	WDIAG1	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE BDiag.Deactivated := TRUE, SDiag := 0 MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	DEACT
31	WDIAG1	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status=NA =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Station_Non_Exist := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	DIAG1
32	WDIAG1	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status=RE/RS/RR/UE/NR =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Invalid_Slave_Response := TRUE BDiag.Station_Non_Exist := FALSE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	DIAG1
33	WDIAG1	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status=DS =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	DIAG1
34	WDIAG1	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Master_Lock = TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag := Diag_Data MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	DIAG1
35	WDIAG1	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Master_Lock = FALSE && Diag_Data.Master_Add <> invalid && SPara.Prm_Data.DPV1_Enable=TRUE && SPara.DPV1_Supported=TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag := Diag_Data MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	UNLCK
36	WDIAG1	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Master_Lock = FALSE && (Diag_Data.Master_Add = invalid SPara.Prm_Data.DPV1_Enable=FALSE SPara.DPV1_Supported=FALSE) =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr SPara.New_Prm := FALSE BDiag := Diag_Data, CHANGE_DIAG_IND MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No_AClr)	PRM
37	PRM	MSCY1M Abort.req (Rem Add) =>ignore	PRM

#	Current State	Event /Condition =>Actions taken&&Output	Next State
38	PRM	MSAL1 Abort.ind (Rem_Add) =>ignore	PRM
39	PRM	MSCY1M Stop Slave Handler.req (Rem Add) =>BDiag.Deactivated := TRUE, SDiag := 0 MSCY1M Stop Slave Handler.cnf(Rem Add)	STOP
40	PRM	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) =>Prm_Data := SPara .Prm_Data Prm_Data.Lock_Req := TRUE Prm_Data.Unlock_Req := FALSE DMPMM1 Set Prm.req (Rem Add, Prm Data)	WPRM
41	PRM	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	PRM
42	PRM	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	PRM
43	PRM	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	PRM
44	WPRM	MSCY1M Abort.req (Rem Add) =>ignore	WPRM
45	WPRM	MSAL1 Abort.ind (Rem_Add) =>ignore	WPRM
46	WPRM	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WPRM
47	WPRM	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WPRM
48	WPRM	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WPRM
49	WPRM	DMPMM1 Set Prm.cnf(+) (Rem Add) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	UNLCK
50	WPRM	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	UNLCK
51	WPRM	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = DS =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	PRM
52	WPRM	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Station_Non_Exist := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG1
53	WPRM	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = RE/RS/RR/UE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Invalid_Slave_Response := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG1
54	WPRM	DMPMM1 Set Prm.cnf(+) (Rem Add) /SPara.Active = TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	CFG
55	CFG	MSCY1M Abort.req (Rem Add) =>ignore	CFG
56	CFG	MSAL1 Abort.ind (Rem_Add) =>ignore	CFG

#	Current State	Event /Condition =>Actions taken&&Output	Next State
57	CFG	MSCY1M Stop Slave Handler.req (Rem Add) =>Prm_Data := SPara .Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	SUNLCK
58	CFG	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) =>Cfg_Data:=SPara .Cfg_Data DMPMM1 Chk Cfg.req (Rem Add, Cfg Data)	WCFG
59	CFG	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	CFG
60	CFG	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	CFG
61	CFG	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	CFG
62	WCFG	MSCY1M Abort.req (Rem Add) =>ignore	WCFG
63	WCFG	MSAL1 Abort.ind (Rem_Add) =>ignore	WCFG
64	WCFG	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WCFG
65	WCFG	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WCFG
66	WCFG	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WCFG
67	WCFG	DMPMM1 Chk Cfg.cnf(+) (Rem Add) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	UNLCK
68	WCFG	DMPMM1 Chk Cfg.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	UNLCK
69	WCFG	DMPMM1 Chk Cfg.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = DS =>Diag:=TRUE, No_AClr:=SPara .Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	CFG
70	WCFG	DMPMM1 Chk Cfg.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA =>Diag:=TRUE, No_AClr:=SPara .Ignore_AClr BDiag.Station_Non_Exist := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG1
71	WCFG	DMPMM1 Chk Cfg.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = RE/RS/RR/UE =>Diag:=TRUE, No_AClr:=SPara .Ignore_AClr BDiag.Invalid_Slave_Response := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG1
72	WCFG	DMPMM1 Chk Cfg.cnf(+) (Rem Add) /SPara.Active = TRUE =>Diag:=TRUE, No_AClr:=SPara .Ignore_AClr MSAL1M_Started := FALSE Delay_Count := SPara .Diag_Upd_Delay Wait_for_Start_con := FALSE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
73	DIAG2	MSCY1M Abort.req (Rem Add) =>GO_ABORT	DIAG2
74	DIAG2	MSAL1 Start.cnf(Rem Add) =>Wait_for_Start_con:=FALSE MSCY1M_Started.ind(Rem_Add)	DIAG2
75	DIAG2	MSAL1 Abort.ind (Rem_Add) =>GO_ABORT	DIAG2

#	Current State	Event /Condition =>Actions taken&&Output	Next State
76	DIAG2	MSCY1M Stop Slave Handler.req (Rem Add) /SPara.DPV1_Supported = FALSE =>Go_Abort := FALSE, BInput := Nil DTrans := 0 Prm_Data := SPara .Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	SUNLCK
77	DIAG2	MSCY1M Stop Slave Handler.req (Rem Add) /SPara.DPV1_Supported = TRUE =>Go_Abort := FALSE, BInput := Nil DTrans := 0 MSAL1 Stop.req (Rem Add)	WSTPCS
78	DIAG2	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && Wait_for_Start_con = FALSE =>DMPMM1 Slave Diag.req (Rem Add)	WDIAG2
79	DIAG2	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && Wait_for_Start_con = TRUE =>Diag:=TRUE, No_AClr:=SPara_.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
80	DIAG2	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort=TRUE && SPara.DPV1_Supported = FALSE =>Go_Abort := FALSE No_AClr := SPara_.Ignore_AClr OR NOT SPara.Active Prm_Data := SPara .Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	WUNLCK
81	DIAG2	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort=TRUE && SPara.DPV1_Supported = TRUE =>Go_Abort := FALSE, Diag := TRUE No_AClr := SPara_.Ignore_AClr OR NOT SPara.Active MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr) MSAL1 Stop.req (Rem Add)	WSTPCC
82	DIAG2	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	DIAG2
83	DIAG2	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	DIAG2
84	DIAG2	MSCY1M Get Input.req (Rem Add, Slot Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	DIAG2
85	WDIAG2	MSCY1M Abort.req (Rem Add) =>GO_ABORT	WDIAG2
86	WDIAG2	MSAL1 Abort.ind (Rem Add) =>GO_ABORT	WDIAG2
87	WDIAG2	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WDIAG2
88	WDIAG2	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WDIAG2
89	WDIAG2	MSCY1M Get Input.req (Rem Add, Slot Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WDIAG2
90	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=FALSE BDiag := Diag_Data, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
91	WDIAG2	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>GO_ABORT, Diag:=FALSE, No_AClr:=FALSE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
92	WDIAG2	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = DS =>Diag:=TRUE, No_AClr:=SPara_.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2

#	Current State	Event /Condition =>Actions taken&&Output	Next State
93	WDIAG2	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA && SPara_NA_To_Abort = FALSE =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr Bdiag.Station_Non_Existent := TRUE SDiag := 1, DTrans := 0 MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
94	WDIAG2	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA && SPara_NA_To_Abort = TRUE =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr, Bdiag.Station_Non_Existent := TRUE, SDiag := 1, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
95	WDIAG2	DMPMM1 Slave Diag.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = RE/RS/RR/UE/NR =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr, Bdiag.Station_Non_Existent := TRUE, Bdiag.Invalid_Slave_Response := TRUE, SDiag := 1, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
96	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && (Diag_Data.Prm_Req = TRUE Diag_Data.Master_Lock = TRUE) && Delay_Count > 0 =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr Delay_Count := Delay_Count-1, Bdiag := Diag_Data BInput := 0, SDiag := 1, DTrans := 0 MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
97	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && (Diag_Data.Prm_Req = TRUE Diag_Data.Master_Lock = TRUE) && Delay_Count = 0 =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr Bdiag := Diag_Data, SDiag := 1, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
98	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Prm_Req = FALSE && Diag_Data.Master_Lock = FALSE && Diag_Data.Station_Not_Ready = TRUE =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr SDiag := 1, DTrans := 0, BInput := Nil Bdiag := Diag_Data MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
99	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Prm_Req = FALSE && Diag_Data.Master_Lock = FALSE && Diag_Data.Station_Not_Ready = FALSE && Diag_Data.Stat_Diag = TRUE =>Diag:=TRUE, No_AClr:=SPara_Ignore_AClr START_MSAL1M, SDiag := 1, DTrans := 0 BInput := Nil, Bdiag := Diag_Data MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
100	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Prm_Req = FALSE && Diag_Data.Master_Lock = FALSE && Diag_Data.Station_Not_Ready = FALSE && Diag_Data.Stat_Diag=FALSE && DTrans = 0 =>Diag:=TRUE, No_AClr:=TRUE SDiag := Diag_Data.Ext_Diag, Delay_Count := 0 Bdiag := Diag_Data, START_MSAL1M, CHANGE_DIAG_IND MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
101	WDIAG2	DMPMM1 Slave Diag.cnf(+) (Rem Add, Diag Data) /SPara.Active = TRUE && Diag_Data.Prm_Req = FALSE && Diag_Data.Master_Lock = FALSE && Diag_Data.Station_Not_Ready = FALSE && Diag_Data.Stat_Diag = FALSE && DTrans = 1 =>Diag:=TRUE, No_AClr:=TRUE SDIAG := Diag_Data.Ext_Diag, CHANGE_DIAG_IND Bdiag := Diag_Data MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
102	DATA	MSCY1M Abort.req (Rem Add) =>GO_ABORT	DATA
103	DATA	MSAL1 Start.cnf (Rem Add) =>Wait_for_Start_con := FALSE, START_MSAL1M MSCY1M_Started.ind(Rem_Add)	DATA
104	DATA	MSAL1 Abort.ind (Rem_Add) =>GO_ABORT	DATA

#	Current State	Event /Condition =>Actions taken&&Output	Next State
105	DATA	MSCY1M Stop Slave Handler.req (Rem Add) /SPara.DPV1_Supported = FALSE =>Go_Abort := FALSE, BInput := Nil DTrans := 0 Prm_Data := SPara .Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	SUNLCK
106	DATA	MSCY1M Stop Slave Handler.req (Rem Add) /SPara.DPV1_Supported = TRUE =>Go_Abort := FALSE, BInput :=Nil, DTrans := 0 MSAL1 Stop.req (Rem Add)	WSTPCS
107	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && Wait_for_Start_con = FALSE && Output_Clear = TRUE && ((SPara.DPV1_Supported = FALSE && SPara.New_Prm = FALSE) SPara.DPV1_Supported = TRUE) && SPara.Fail_Safe = FALSE =>Outp_Data:=0 DMPMM1 Data Exchange.req (Rem Add, Outp Data)	WDATA
108	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && Wait_for_Start_con = FALSE && Output_Clear = TRUE && ((SPara.DPV1_Supported = FALSE && SPara.New_Prm = FALSE) SPara.DPV1_Supported = TRUE) && SPara.Fail_Safe = TRUE =>Outp_Data.len:=0 DMPMM1 Data Exchange.req (Rem Add, Outp Data)	WDATA
109	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && Wait_for_Start_con = FALSE && Output_Clear = FALSE && ((SPara.DPV1_Supported = FALSE && SPara.New_Prm = FALSE) SPara.DPV1_Supported = TRUE) =>Outp_Data:=BOutput DMPMM1 Data Exchange.req (Rem Add, Outp Data)	WDATA
110	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && SPara.DPV1_Supported = FALSE && SPara.New_Prm = TRUE =>Prm_Data := SPara .Prm_Data Prm_Data.Lock_Req := TRUE Prm_Data.Unlock_Req := FALSE DMPMM1 Set Prm.req (Rem Add, Prm Data)	WDATA
111	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort = FALSE && Wait_for_Start_con = TRUE =>Diag:=TRUE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
112	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort:=TRUE && SPara.DPV1_Supported = FALSE =>Go_Abort := FALSE, Prm_Data:= SPara .Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	WUNLCK
113	DATA	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) /Go_Abort:=TRUE && SPara.DPV1_Supported = TRUE =>Go_Abort := FALSE, Diag := TRUE No_AClr := SPara .Ignore_AClr OR NOT SPara.Active MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr) MSAL1 Stop.req (Rem Add)	WSTPCC
114	DATA	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	DATA
115	DATA	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	DATA
116	DATA	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	DATA
117	WDATA	MSCY1M Abort.req (Rem Add) =>GO_ABORT	WDATA
118	WDATA	MSAL1 Abort.ind (Rem_Add) =>GO_ABORT	WDATA
119	WDATA	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WDATA

#	Current State	Event /Condition =>Actions taken&&Output	Next State
120	WDATA	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WDATA
121	WDATA	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WDATA
122	WDATA	DMPMM1 Data Exchange.cnf(+) (Rem Add, Diag Flag, Inp Data) /SPara.Active = FALSE =>GO_ABORT, Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
123	WDATA	DMPMM1 Data Exchange.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>GO_ABORT, Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
124	WDATA	DMPMM1 Data Exchange.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = DS =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr DTrans := 0, BInput := Nil MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
125	WDATA	DMPMM1 Data Exchange.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA && SPara_NA_To_Abort = FALSE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Station_Non_Existent := TRUE, SDiag := 1 MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
126	WDATA	DMPMM1 Data Exchange.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status=NA && SPara_NA_To_Abort = TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Station_Non_Existent := TRUE, SDiag := 1 GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
127	WDATA	DMPMM1 Data Exchange.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = RE/RS/RR/UE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Invalid_Slave_Response := TRUE BDiag.Station_Non_Existent := FALSE SDiag := 1, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
128	WDATA	DMPMM1 Data Exchange.cnf(+) (Rem Add, Diag Flag, Inp Data) /SPara.Active = TRUE && Diag_Flag = FALSE && Inp_Data.len = Exp_Inp_Len =>Diag:=FALSE, No_AClr:=TRUE BDiag.Station_Non_Existent := FALSE BInput := Inp_Data, DTrans := 1 SDiag := BDiag.Ext_Diag MSCY1M_New_Input.ind(Rem_Add) MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
129	WDATA	DMPMM1 Data Exchange.cnf(+) (Rem Add, Diag Flag, Inp Data) /SPara.Active = TRUE && Inp_Data.len <> Exp_Inp_Len && (Inp_Data.len <> 1 Exp_Inp_Len <> 0 Diag_Flag = FALSE) =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Invalid_Slave_Response := TRUE BDiag.Station_Non_Existent := FALSE SDiag := 1, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
130	WDATA	DMPMM1 Data Exchange.cnf(+) (Rem Add, Diag Flag, Inp Data) /SPara.Active = TRUE && Diag_Flag = TRUE && Inp_Data.len = Exp_Inp_Len =>Diag:=FALSE, No_AClr:=TRUE BDiag.Station_Non_Existent := FALSE BInput := Inp_Data, DTrans := 1 SDiag := BDiag.Ext_Diag MSCY1M_New_Input.ind(Rem_Add) MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2
131	WDATA	DMPMM1 Data Exchange.cnf(+) (Rem Add, Diag Flag, Inp Data) /SPara.Active = TRUE && Diag_Flag = TRUE && Inp_Data.len = 1 && Exp_Inp_Len = 0 =>Diag:=FALSE, No_AClr:=TRUE BDiag.Station_Non_Existent := FALSE, DTrans := 1 SDiag := BDiag.Ext_Diag MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG2

#	Current State	Event /Condition =>Actions taken&&Output	Next State
132	WDATA	DMPMM1 Set Prm.cnf(+) (Rem Add) /SPara.Active = FALSE =>GO_ABORT, Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
133	WDATA	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>GO_ABORT, Diag:=FALSE, No_AClr:=TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
134	WDATA	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = DS =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
135	WDATA	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA && SPara.NA_To_Abort = FALSE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Station_Non_Existent := TRUE, SDiag := 1 MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
136	WDATA	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = NA && SPara.NA_To_Abort = TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Station_Non_Existent := TRUE, SDiag := 1 GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
137	WDATA	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE && Status = RE/RS/RR/UE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr BDiag.Invalid_Slave_Response := TRUE BDiag.Station_Non_Existent := FALSE SDiag := 1, GO_ABORT MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
138	WDATA	DMPMM1 Set Prm.cnf(+) (Rem Add) /SPara.Active = TRUE =>Diag:=TRUE, No_AClr:=TRUE SPara.New_Prm := FALSE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DATA
139	UNLCK	MSCY1M Abort.req (Rem Add) =>ignore	UNLCK
140	UNLCK	MSAL1 Abort.ind (Rem_Add) =>ignore	UNLCK
141	UNLCK	MSCY1M Stop Slave Handler.req (Rem Add) =>Prm_Data := SPara.Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	SUNLCK
142	UNLCK	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) =>Prm_Data := SPara.Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	WUNLCK
143	UNLCK	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	UNLCK
144	UNLCK	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	UNLCK
145	UNLCK	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	UNLCK
146	WUNLCK	MSCY1M Abort.req (Rem Add) =>ignore	WUNLCK
147	WUNLCK	MSAL1 Abort.ind (Rem_Add) =>ignore	WUNLCK
148	WUNLCK	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WUNLCK
149	WUNLCK	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WUNLCK

#	Current State	Event /Condition =>Actions taken&&Output	Next State
150	WUNLCK	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WUNLCK
151	WUNLCK	DMPMM1 Set Prm.cnf(+) (Rem Add) /SPara.Active = TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG1
152	WUNLCK	DMPMM1 Set Prm.cnf(+) (Rem Add) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE, SDiag := 0 BDiag.Deactivated := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DEACT
153	WUNLCK	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = TRUE =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DIAG1
154	WUNLCK	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) /SPara.Active = FALSE =>Diag:=FALSE, No_AClr:=TRUE, SDiag := 0 BDiag.Deactivated := TRUE MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	DEACT
155	SUNLCK	MSCY1M Abort.req (Rem Add) =>ignore	SUNLCK
156	SUNLCK	MSAL1 Abort.ind (Rem_Add) =>ignore	SUNLCK
157	SUNLCK	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	SUNLCK
158	SUNLCK	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	SUNLCK
159	SUNLCK	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	SUNLCK
160	SUNLCK	DMPMM1 Set Prm.cnf(+) (Rem Add) =>SDiag := 0, BDiag.Deactivated := TRUE MSCY1M Stop Slave Handler.cnf(Rem Add)	STOP
161	SUNLCK	DMPMM1 Set Prm.cnf(-) (Rem Add, Status) =>SDiag := 0, BDiag.Deactivated := TRUE MSCY1M Stop Slave Handler.cnf(Rem Add)	STOP
162	WSTPCC	MSAL1 Start.cnf(Rem Add) =>MSAL1M_Started := FALSE	WSTPCC
163	WSTPCC	MSAL1 Stop.cnf(Rem Add) /MSAL1M_Started = TRUE =>MSCY1M Stopped.ind (Rem Add)	UNLCK
164	WSTPCC	MSAL1 Stop.cnf(Rem Add) /MSAL1M_Started = FALSE =>ignore	UNLCK
165	WSTPCC	MSCY1M Cont Slave Handler.req (Rem Add, Output Clear) =>Diag:=TRUE, No_AClr:=SPara.Ignore_AClr OR NOT SPara.Active MSCY1M Cont Slave Handler.cnf(Rem Add, Diag, No AClr)	WSTPCC
166	WSTPCC	MSCY1M Abort.req (Rem Add) =>ignore	WSTPCC
167	WSTPCC	MSAL1 Abort.ind (Rem_Add) =>ignore	WSTPCC
168	WSTPCC	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WSTPCC
169	WSTPCC	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WSTPCC
170	WSTPCC	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WSTPCC

#	Current State	Event /Condition =>Actions taken&&Output	Next State
171	WSTPCS	MSAL1 Start.cnf (Rem Add) =>MSAL1M_Started := FALSE	WSTPCS
172	WSTPCS	MSAL1 Stop.cnf (Rem Add) /MSAL1M_Started = TRUE =>Prm_Data := SPara .Prm_Data Prm_Data.Unlock_Req := TRUE MSCY1M Stopped.ind (Rem Add) DMPMM1 Set Prm.req (Rem Add, Prm Data)	SUNLCK
173	WSTPCS	MSAL1 Stop.cnf (Rem Add) /MSAL1M_Started = FALSE =>Prm_Data := SPara .Prm_Data Prm_Data.Unlock_Req := TRUE DMPMM1 Set Prm.req (Rem Add, Prm Data)	SUNLCK
174	WSTPCS	MSCY1M Abort.req (Rem Add) =>ignore	WSTPCS
175	WSTPCS	MSAL1 Abort.ind (Rem_Add) =>ignore	WSTPCS
176	WSTPCS	MSCY1M Get Slave Diag.req (Rem Add) =>Diag_Data := BDiag MSCY1M Get Slave Diag.cnf(+) (Rem Add, Diag Data)	WSTPCS
177	WSTPCS	MSCY1M Set Output.req (Rem Add, Slot Number, Output Data) =>BOutput(Slot_Number) := Output_Data MSCY1M Set Output.cnf(+) (Rem Add, Slot Number)	WSTPCS
178	WSTPCS	MSCY1M Get Input.req (Rem Add, Slot_Number) =>Input_Data := BInput(Slot_Number) MSCY1M Get Input.cnf(+) (Rem Add, Input Data)	WSTPCS
179	ANY-STATE	MSCY1M Reset.req (Rem Add) =>MSCY1M Reset.cnf (Rem Add)	POWER-ON

6.6.6 MSAL1M

6.6.6.1 Primitive Definitions

6.6.6.1.1 Primitives Exchanged between FSPMM1 and MSAL1M

Table 408 - Primitives issued by FSPMM1 to MSAL1M

Primitive Name	Source	Associated Parameters	Functions
MInit.req	FSPMM1	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMM1	(none)	
Abort.req	FSPMM1	(none)	

Table 409 - Primitives issued by MSAL1M to FSPMM1

Primitive Name	Source	Associated Parameters	Functions
MInit.cnf	MSAL1M	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MSAL1M	(none)	
Abort.ind	MSAL1M	(none)	
Fault.ind	MSAL1M	(none)	
Started.ind	MSAL1M	Alarm Limit	
Stopped.ind	MSAL1M	(none)	
Alarm Notification.ind	MSAL1M	Slot Number, Alarm Type, Seq Nr, Add Ack, Alarm Specifier, Alarm Data	

6.6.6.1.2 Primitives Exchanged between MSCY1M and MSAL1M

Table 410 - Primitives issued by MSCY1M to MSAL1M

Primitive Name	Source	Associated Parameters	Functions
Start.req	MSCY1M	Rem Add	Start MS1-processing
Stop.req	MSCY1M	Rem Add	Stop MS1-processing
Change Diag.req	MSCY1M	Rem Add, Diag Data	Indicates change of diagnosis

Table 411 - Primitives issued by MSAL1M to MSCY1M

Primitive Name	Source	Associated Parameters	Functions
Abort.ind	MSAL1M	Rem Add	MSAL1 detected an error
Start.cnf	MSAL1M	Rem Add	Start completed
Stop.cnf	MSAL1M	Rem Add	Stop completed

6.6.6.1.3 Parameters of MSAL1M Primitives

Table 412 – Parameter Used with Primitives Exchanged between MSAL1M and MSCY1M

Parameter Name	Description
Rem Add	This parameter conveys the DL address of the assigned DP-Slave.
Diag Data	This parameter conveys the new diagnosis information of a DP-Slave

6.6.6.2 State Machine Description

The Alarm State Machine of a DP-Master (Class 1) handles alarm messages, which have been indicated by a DP-Slave.

When started the State Machine waits for being started by MSCY1M. The Start.req follows, when the assigned DP-Slave has reached the state DATA-EXCH. MSAL1M will be stopped by MSCY1M if the corresponding DP-Slave leaves the state DATA-EXCH.

The service Slave_Diag.cnf provides new Diag_Data to MSCY1M. MSCY1M copies the diagnosis information into the User data interface. Additionally MSCY1M hands over an Alarm in Ext_Diag_Data to MSAL1M.

Alarm-Requests are indicated to the User. The User has to confirm the Alarm-Indications with the service Alarm_Ack.

The MSAC1 State Machine indicates the termination of the alarm sequence with the service primitive Alarm_Ack.cnf.

Two types of alarm handling are provided by a DP-Master (Class 1) the type mode and the sequence mode. Depending on the chosen mode the number of outstanding alarms per DP-Slave differs. In type mode one pending alarm per type and per DP-Slave is allowed. 6 different alarm types are defined. In sequence mode the alarm type is irrelevant. Only the number of simultaneously processed alarms for each DP-Slave is limited to a value between 2 and 32.

The State Machine needs some local variables which are described in clause a). Furthermore the machine uses functions and macros which are listed in clause b) and c).

a) Local variables of the MSAL1M

Alarm_Sequence

(Boolean)

This variable indicates whether:

only one alarm of a specific alarm type can be active at one time (type mode: Alarm_Sequence=False)
or

several alarms (2 to 32) of any type can be active at one time (sequence mode: Alarm_Sequence=True).

Alarm_State_Table

(Unsigned8)

The Alarm_State_Table is a two dimensional array with 7 * 32 elements. Each element within the array stores information about the actual state of any alarm. The array is indexed with the alarm class calculated from the Alarm_Type and the Seq_Nr of the alarm.

Range**Table 413 - Possible Values in the Alarm_State_Table**

idle	No service is being processed according to the actual index.
w_res	Wait for MSAL1_Alarm.res according to the actual index.
w_req	For this index (Alarm_Type,Seq_Nr) an Alarm_Acknowledge is stored in the Alarm_Ack_FIFO and MSAL1M waits for another MSAC1M_Alarm_Ack.cnf.
w_con	Wait for MSAC1M_Alarm_Ack.cnf according to the actual index.

Alarm_Limit

(Unsigned8)

This variable indicates the maximum number of parallel alarms of each Slave.

Range: 7..Bus_Para.Alarm_Max

Alarm_Decompose

(Array 0..7 of Unsigned8)

Table for Decoding the number of parallel alarms.

Range:

Alarm_Decompose[0]= 1

Alarm_Decompose[1]= 2

Alarm_Decompose[2]= 4

Alarm_Decompose[3]= 8

Alarm_Decompose[4]=12

Alarm_Decompose[5]=16

Alarm_Decompose[6]=24

Alarm_Decompose[7]=32

Actual_Max_Alarm_Len

(Unsigned8)

Actual_Max_Alarm_Len contains the actual maximum length of an Alarm.

Range: 4..64

Alarm_Count

(Unsigned8)

This counter contains the number of alarms which have actually been sent by the Slave. The counter is only used in the sequence mode.

Range: 0..32

Waiting_For_Alarm_Ack

(Boolean)

This variable shows, whether the Alarm State Machine is actually waiting for the confirmation of any alarm acknowledge (True) or not (False).

Alarm_Ack_FIFO

The Alarm_Ack_FIFO is used to store the Alarm_Ack which are still to be sent. Each element of this alarm acknowledge FIFO has 3 Unsigned8 entries: Alarm_Type, Slot_Number and Seq_Nr. The FIFO must be able to hold up to 32 respectively Bus_Para.Alarm_Max entries.

Alarm_Ack_FIFO_Filled

(Boolean)

This variable indicates whether the Alarm_Ack_FIFO contains at least one entry (True) or it is empty (False). The value of the variable is adapted with each access to the FIFO.

Act_Alarm_PDU

(Octet-String)

The structure Act_Alarm_PDU is temporarily used to store one Alarm while being evaluated.

Stored_Alarm_PDU

(Octet-String)

The structure Stored_Alarm_PDU is used to store exactly one Alarm-PDU. Up to one Alarm-PDU must be saved locally during waiting for MSAC1M_Alarm_Ack enf of the according Seq_Nr, because only one channel for alarm acknowledges exists.

Actual_Enabled_Alarms

(Bitarea[0..7])

This variable indicates the type of alarms which are actually supported by the Master.

Bit 0 = reserved

Bit 1 = reserved

Bit 2 = Update_Alarm

Bit 3 = Status_Alarm

Bit 4 = Manufacturer_Specific_Alarm

Bit 5 = Diagnostic_Alarm

Bit 6 = Process_Alarm

Bit 7 = Pull_Plug_Alarm

b) Functions

Fill_Alarm_State_Table(State)

This function sets each entry of the two dimensional Alarm_State_Table to the given initial value of State. Valid parameters for State are idle, w_res, w_req and w_con. The function has no return value.

Reset_Alarm_Ack_FIFO()

This function resets the FIFO for alarm acknowledges. During this function all entries are cleared and the variable Alarm_Ack_FIFO_Filled is thereby set to False. The function has no return value.

Store_To_Alarm_Ack_FIFO(Alarm_Type, Slot_Number, Seq_Nr)

By means of this function one set of the 3 alarm recognition parameters Alarm_Type, Slot_Number and Seq_Nr is stored from the given parameters into the Alarm_Ack_FIFO. The function has no return value.

Load_From_Alarm_Ack_FIFO(Alarm_Type, Slot_Number, Seq_Nr)

By means of this function one set of the 3 alarm recognition parameters Alarm_Type, Slot_Number and Seq_Nr is read from the Alarm_Ack_FIFO and stored into the given parameters. The function has no return value.

AcIs(Alarm_Type)

This function calculates the Alarm_Type related index as return value as follows:

```

IF (Alarm_Type = 1) THEN return 0
IF (Alarm_Type = 2) THEN return 1
IF (Alarm_Type = 3) THEN return 2
IF (Alarm_Type = 4) THEN return 3
IF (Alarm_Type = 5) THEN return 4
IF (Alarm_Type = 6) THEN return 5
IF (Alarm_Type >= 32) && (Alarm_Type <= 126) THEN return 6

```

c) Macros**ALARM_ENABLED**

```

(
  ((Alarm_Type=3 OR Alarm_Type=4) &&
  Actual_Enabled_Alarms[7]=TRUE)
  OR ((Alarm_Type=2) && Actual_Enabled_Alarms[6]=TRUE)
  OR ((Alarm_Type=1) && Actual_Enabled_Alarms[5]=TRUE)
  OR ((Alarm_Type>=32 && Alarm_Type<=126)
  && Actual_Enabled_Alarms[4]=TRUE)
  OR ((Alarm_Type=5) && Actual_Enabled_Alarms[3]=TRUE)
  OR ((Alarm_Type=6) && Actual_Enabled_Alarms[2]=TRUE)
)

```

6.6.6.3 MSAL1M State Table**Table 414 - MSAL1M State Table**

#	Current State	Event / Condition =>Actions taken&&Output	Next State
1	POWER-ON	MSAL1M_Minit_MS1.req(Rem_Add) =>MSAL1M_Minit_MS1.cnf(Rem_Add)	WSTART
2	POWER-ON	MSAL1M_Abort.req(Rem_Add) =>ignore	POWER-ON
3	POWER-ON	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) =>MSAL1M_Alarm_Ack.cnf(-)(Rem_Add, Status:=Not_Initialized)	POWER-ON
4	WSTART	MSCY1M_Start.req(Rem_Add) /Alarm_Mode = 0 =>Alarm_Sequence := False Alarm_Limit := 7 Actual_Enabled_Alarms := Enabled_Alarms Actual_Max_Alarm_Len := Max_Alarm_Len Fill_Alarm_State_Table(idle) Reset_Alarm_Ack_FIFO() Alarm_Count := 0 Clear_Stored_Alarm_PDU Waiting_For_Alarm_Ack := False MSAL1M_Started.ind(Rem_Add, Alarm_Limit) MSCY1M_Start.cnf(Rem_Add)	W-DIA-EVENT

#	Current State	Event /Condition =>Actions taken&&Output	Next State
5	WSTART	MSCY1M_Start.req(Rem_Add) /Alarm_Mode <> 0 =>Alarm_Sequence := True Alarm_Limit := min(Bus_Para.Alarm_Max, Alarm_Decode[Alarm_Mode]) Actual_Enabled_Alarms := Enabled_Alarms Actual_Max_Alarm_Len := Max_Alarm_Len Fill_Alarm_State_Table(idle) Reset_Alarm_Ack_FIFO() Alarm_Count := 0 Clear_Stored_Alarm_PDU Waiting_For_Alarm_Ack := False MSAL1M_Started.ind(Rem_Add, Alarm_Limit) MSCY1M_Start.cnf(Rem_Add)	W-DIA-EVENT
6	WSTART	MSCY1M_Stop.req(Rem_Add) =>MSAL1M_Fault.ind	POWER-ON
7	WSTART	MSCY1M_Change_Diag.req(Rem_Add, Diag_Block) =>MSAL1M_Fault.ind	WSTART
8	WSTART	MSAL1M_Abort.req(Rem_Add) =>ignore	WSTART
9	WSTART	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) =>MSAL1M_Alarm_Ack.cnf(-)(Rem_Add, Status:=Not_Started)	WSTART
10	WSTART	MSAC1M_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) =>MSAL1M_Fault.ind	POWER-ON
11	W-DIA-EVENT	MSAL1M_Minit_MS1.req(Rem_Add) =>MSAL1M_Fault.ind	POWER-ON
12	W-DIA-EVENT	MSAL1M_Abort.req(Rem_Add) =>MSAL1M_Abort.ind(Rem_Add)	WSTART
13	W-DIA-EVENT	MSCY1M_Start.req(Rem_Add) =>MSAL1M_Fault.ind	POWER-ON
14	W-DIA-EVENT	MSCY1M_Stop.req(Rem_Add) =>MSAL1M_Stopped.ind(Rem_Add) MSCY1M_Stop.cnf(Rem_Add)	WSTART
15	W-DIA-EVENT	MSCY1M_Change_Diag.req(Rem_Add, Diag_Block) =>Act_Alarm_PDU := Alarm_PDU	CHK-DIA-ALARM
16	W-DIA-EVENT	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) /ALARM_ENABLED(Alarm_Type) = False =>MSAL1M_Alarm_Ack.cnf(-)(Rem_Add, Status:=Not_Enabled)	W-DIA-EVENT
17	W-DIA-EVENT	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) /ALARM_ENABLED(Alarm_Type) = True && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] <> w_res =>MSAL1M_Alarm_Ack.cnf(-)(Rem_Add, Status:=Alarm_Not_Pending)	W-DIA-EVENT
18	W-DIA-EVENT	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) /ALARM_ENABLED(Alarm_Type) = True && Alarm_Sequence = False && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] = w_res && Waiting_For_Alarm_Ack = False =>Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] := w_con Waiting_For_Alarm_Ack := True MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr)	W-DIA-EVENT
19	W-DIA-EVENT	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) /ALARM_ENABLED(Alarm-PDU.Alarm_Type) = True && Alarm_Sequence = False && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] = w_res && Waiting_For_Alarm_Ack = True =>Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] := w_req Store_To_Alarm_Ack_FIFO(Alarm_Type, Slot_Number, Seq_Nr)	W-DIA-EVENT
20	W-DIA-EVENT	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) /ALARM_ENABLED(Alarm_Type) = True && Alarm_Sequence = True && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] = w_res && Waiting_For_Alarm_Ack = False =>Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] := w_con Alarm_Count := Alarm_Count-1 Waiting_For_Alarm_Ack := True MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr)	W-DIA-EVENT

#	Current State	Event /Condition =>Actions taken&&Output	Next State
21	W-DIA-EVENT	MSAL1M_Alarm_Ack.req(Rem_Add, Alarm_Type, Slot_Number, Seq_Nr) /ALARM_ENABLED(Alarm_Type) = True && Alarm_Sequence = True && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] = w_res && Waiting_For_Alarm_Ack = True =>Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] := w_req Store_To_Alarm_Ack_FIFO(Alarm_Type, Slot_Number, Seq_Nr)	W-DIA-EVENT
22	W-DIA-EVENT	MSAC1M_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Waiting_For_Alarm_Ack = False =>MSAL1M_Fault.ind	POWER-ON
23	W-DIA-EVENT	MSAC1M_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Waiting_For_Alarm_Ack = True && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] <> w_con =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
24	W-DIA-EVENT	MSAC1M_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Waiting_For_Alarm_Ack = True && Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] = w_con =>Alarm_State_Table[Acls(Alarm_Type), Seq_Nr] := idle Waiting_For_Alarm_Ack := False MSAL1M_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr)	SEND-NEXT-ALARM
25	CHK-DIA-ALARM	/Act_Alarm_PDU not present =>ignore	W-DIA-EVENT
26	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = False =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
27	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = False && Alarm_State_Table[Acls(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = w_res/w_req =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
28	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = False && Alarm_State_Table[Acls(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = w_con && Stored_Alarm_PDU present =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
29	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = False && Alarm_State_Table[Acls(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = idle =>Alarm_State_Table[Acls(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] := w_res MSAL1M_Alarm_Notification.ind(Rem_Add, Alarm_Type:=Act_Alarm_PDU.Alarm_Type, Slot_Number:=Act_Alarm_PDU.Slot_Number, Seq_Nr:=Act_Alarm_PDU.Seq_Nr, Alarm_Specifier:=Act_Alarm_PDU.Alarm_Specifier, Add_Ack:=Act_Alarm_PDU.Add_Ack, Alarm_Data:=Act_Alarm_PDU.Diagnostic_User_Data)	W-DIA-EVENT
30	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = False && Alarm_State_Table[Acls(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = w_con && Stored_Alarm_PDU not present =>Stored_Alarm_PDU := Act_Alarm_PDU	W-DIA-EVENT

#	Current State	Event /Condition =>Actions taken&&Output	Next State
31	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = True && Alarm_Count >= Alarm_Limit =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
32	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = True && Alarm_Count < Alarm_Limit && Alarm_State_Table[AcIs(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = w_res/w_req =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
33	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = True && Alarm_Count < Alarm_Limit && Alarm_State_Table[AcIs(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = idle =>Alarm_State_Table[AcIs(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] := w_res Alarm_Count := Alarm_Count+1 MSAL1M_Alarm_Ack.ind(Rem_Add, Alarm_Type:=Act_Alarm_PDU.Alarm_Type, Slot_Number:=Act_Alarm_PDU.Slot_Number, Seq_Nr:=Act_Alarm_PDU.Seq_Nr, Alarm_Specifier:=Act_Alarm_PDU.Alarm_Specifier, Add_Ack:=Act_Alarm_PDU.Add_Ack, Alarm_Data:=Act_Alarm_PDU.Diagnostic_User_Data)	W-DIA-EVENT
34	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = True && Alarm_Count < Alarm_Limit && Alarm_State_Table[AcIs(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = w_con && Stored_Act_Alarm_PDU present =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
35	CHK-DIA-ALARM	/Act_Alarm_PDU present && ALARM_ENABLED(Act_Alarm_PDU.Alarm_Type) = True && Alarm_Sequence = True && Alarm_Count < Alarm_Limit && Alarm_State_Table[AcIs(Act_Alarm_PDU.Alarm_Type), Act_Alarm_PDU.Seq_Nr] = w_con && Stored_Alarm_PDU not present =>Stored_Alarm_PDU := Act_Alarm_PDU Alarm_Count := Alarm_Count+1	W-DIA-EVENT
36	SEND-NEXT-ALARM	/Stored_Alarm_PDU not present =>ignore	READ-NEXT-ACK
37	SEND-NEXT-ALARM	/Stored_Alarm_PDU present =>Alarm_State_Table[AcIs(Stored_Alarm_PDU.Alarm_Type), Stored_Alarm_PDU.Seq_Nr] := w_res Clear Stored_Alarm_PDU MSAL1M_Alarm_Notification.ind(Rem_Add, Alarm_Type:=Stored_Alarm_PDU.Alarm_Type, Slot_Number:=Stored_Alarm_PDU.Slot_Number, Seq_Nr:=Stored_Alarm_PDU.Seq_Nr, Alarm_Specifier:=Stored_Alarm_PDU.Alarm_Specifier, Add_Ack:=Stored_Alarm_PDU.Add_Ack, Alarm_Data:=Stored_Alarm_PDU.Diagnostic_User_Data)	READ-NEXT-ACK
38	READ-NEXT-ACK	/Alarm_Ack_FIFO_Filled = False =>ignore	W-DIA-EVENT

#	Current State	Event /Condition =>Actions taken&&Output	Next State
39	READ-NEXT-ACK	/Alarm_Ack_FIFO_Filled = True =>Load_From_Alarm_Ack_FIFO(New_Alarm_Type, New_Slot_Number, New_Seq_Nr)	SEND-NEXT-ACK
40	SEND-NEXT-ACK	/Alarm_State_Table[Acls(New_Alarm_Type), New_Seq_Nr] <> w_req =>MSAL1M_Abort.ind(Rem_Add) MSAL1M_Stopped.ind(Rem_Add)	WSTART
41	SEND-NEXT-ACK	/Alarm_State_Table[Acls(New_Alarm_Type), New_Seq_Nr] = w_req =>Alarm_State_Table[Acls(New_Alarm_Type), New_Seq_Nr] := w_con Alarm_Count := Alarm_Count-1 Waiting_For_Alarm_Ack := True MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number:=New_Slot_Number, Alarm_Type:=New_Alarm_Type, Seq_Nr:=New_Seq_Nr)	W-DIA-EVENT
42	ANY-STATE	MSAL1M_Reset.req(Rem_Add) =>MSAL1M_Reset.cnf(Rem_Add)	POWER-ON

6.6.7 MSAC1M

6.6.7.1 Primitive Definitions

6.6.7.1.1 Primitives Exchanged between FSPMM1 and MSAC1M

Table 415 - Primitives issued by FSPMM1 to MSAC1M

Primitive Name	Source	Associated Parameters	Functions
Read.req	FSPMM1	Rem Add, Slot Number, Index, Length	
Write.req	FSPMM1	Rem Add, Slot Number, Index, Length, Data	
Alarm Ack.req	FSPMM1	Rem Add, Slot Number, Alarm Type, Seq Nr	

Table 416 - Primitives issued by MSAC1M to FSPMM1

Primitive Name	Source	Associated Parameters	Functions
Reject.ind	MSAC1M	Rem Add, Status	
Read.cnf(+)	MSAC1M	Rem Add, Length, Data	
Read.cnf(-)	MSAC1M	Rem Add, Error Decode, Error Code 1 Error Code 2	
Write.cnf(+)	MSAC1M	Rem Add, Length	
Write.cnf(-)	MSAC1M	Rem Add, Error Decode, Error Code 1 Error Code 2	
Alarm Ack.cnf(+)	MSAC1M	Rem Add, Slot Number, Alarm Type, Seq Nr	
Alarm Ack.cnf(-)	MSAC1M	Rem Add, Slot Number, Alarm Type, Seq Nr	

6.6.7.1.2 Primitives Exchanged between MSAC1M and MSAL1M

Table 417 - Primitives issued by MSAL1M to MSAC1M

Primitive Name	Source	Associated Parameters	Functions
SInit MSAC1.req	MSAL1M	Rem Add	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	MSAL1M	Rem Add	
Start.req	MSAL1M	Rem Add	Start MS1-processing
Stop.req	MSAL1M	Rem Add	Stop MS1-processing

Table 418 - Primitives issued by MSAC1M to MSAL1M

Primitive Name	Source	Associated Parameters	Functions
SInit MSAC1.cnf	MSAC1M	Rem Add	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MSAC1M	Rem Add	
Start.cnf	MSAC1M	Rem Add	Start completed
Stop.cnf	MSAC1M	Rem Add	Stop completed
Fault.ind	MSAC1M	(none)	
Abort.ind	MSAC1M	Rem Add	MSAC1 detected an error

6.6.7.1.3 Parameters of MSAL1M Primitives

Table 419 – Parameter Used with Primitives Exchanged between MSAL1M and MSCY1M

Parameter Name	Description
Rem Add	This parameter conveys the DL address of the assigned DP-Slave.

6.6.7.2 State Machine Description

The MSAC1M State Machine has the three states POWER-ON, CLOSED and OPEN. With the Init.req from the FSPMM1, the State Machine is initialised and changes to CLOSED. The state transition from CLOSED to OPEN is caused by the Start.req from the Slave-Handler (MSCY1M). The Slave-Handler is calling this service as soon as the DP-Slave has entered the DATA-EXCH-state. When the DP-Slave leaves the DATA-EXCH-state, the Stop.req service is called and the MSAC1M State Machine changes from OPEN to CLOSED. A Read/_Write request with an illegal parameter or a parallel service request while the last Read/Write is still being processed, is rejected to the User with a Reject.ind. An Alarm_Ack.req containing illegal service parameters or a parallel Alarm_Ack.req while the last Alarm_Ack.req is still being processed, is causing a state transition to POWER-ON and a Fault.ind to the FSPMM1.

The MSAC1M State Machine takes into account that the Alarm_Ack can be handled either on the SAP 51 as the Read/Write-service or on SAP 50.

MSAC1M_Alarm_Ack is handled on the same SAP as MSAC1M_Read/Write

If a MSAC1M_Alarm_Ack is requested by the MSAL1M State Machine or a Read/_Write is requested by the User while no service is being processed, the service request is passed directly to the DL. If a Alarm_Ack.req is requested and a Read/Write is being processed, the Alarm_Ack.req is stored and sent as soon as the confirmation of the Read/Write is received. When a Read/Write.req is requested and a Alarm_Ack is being processed, the Read/Write is stored and sent as soon as the confirmation of the Alarm_Ack is received.

When recognising an error while a service request is being processed, the MSAC1M State Machine

- aborts the MS0 connection by indicating a Abort.ind to the MSCY1S,
- deletes an eventually stored service,
- informs the User with a Reject.ind in case of an outstanding Read/_Write service
- and changes from state OPEN to CLOSE.

When a Stop.req is requested by the Slave-Handler, the MSAC1M first waits for an eventually outstanding DATA-REPLY.cnf. After this the Stop.req will be processed. A Read/_Write-confirmation is passed to the User and a possibly stored Alarm_Ack.req is deleted. In case of an Alarm_Ack confirmation, this confirmation is ignored, and a possibly outstanding Read/_Write is deleted and the

User is informed with a Reject.ind. After that, the Stop.cnf is passed to the Slave-Handler and the MSAC1M State Machine changes from state OPEN to CLOSE.

MSAC1M_Alarm_Ack is handled on SAP 50

When a Alarm_Ack is requested by the MSAL1M State Machine or a Read/_Write service is requested by the User, the service request is passed directly to the DL.

When recognising an error while a service request is being processed on the other SAP, the MSAC1M State Machine

waits for the confirmation of the service on the other SAP

then aborts the MSCY1M connection by indicating a Abort.ind to the Slave-Handler MSCY1S,

informs the User with a Reject.ind in case of an outstanding _Read/ _Write service

and changes from state OPEN to CLOSE.

When a Stop.req is requested by the Slave-Handler, the MSAC1M first waits for eventually outstanding DL-DATA-REPLY.cnf. After this the Stop.req will be processed. An outstanding MSAC1M_Read/_Write confirmation is passed to the User and an outstanding Alarm_Ack confirmation is ignored. Then the Stop.cnf is passed to the Slave-Handler and the MSAC1M State Machine changes from state OPEN to CLOSE.

The State Machine needs some local variables which are described in clause a). Furthermore the machine uses functions and macros which are listed in clause b) and c).

a) Local variables of the MSAC1M

Go_Abort

(Boolean)

This variable is set TRUE if the Stop.req is in use.

Internal_Abort

(Boolean)

This variable is set TRUE if an error in the Slave response occurs.

Max_Data_Length

(Unsigned8)

This variable defines the maximum size of MS1-PDU for all DP-Slaves.

Alarm_DSAP

(Unsigned8)

This variable contains the DSAP the Alarm_Ack is sent to.

Service_Header

(Unsigned8)

This variable contains the Function_Num of the MS1-Service actually sent to the DSAP 51.

Service_Buffer

(Octet-String)

This variable contains a Read or Write Request-PDU if a Alarm_Ack is in use (DSAP 51).

Alarm_Buffer

(Octet-String)

This variable contains a Alarm_Ack if a Read or Write Request-PDU is in use (DSAP 51).

(Unsigned8)

Arc

(Unsigned8)

This variable contains the number of the actual Alarm_Ack Services (only 0 or 1 is possible)

Stop_Pending

(Boolean)

This variable is set TRUE if an Stop.cnf is pending.

b) Macros and Replacements

<REQPARAM>

```
<
XXX=Read      : Slot_Number, Index, Length
XXX=Write     : Slot_Number, Index, Length, Data
>
```

<VALID SERVICE FUNCTION NUM>

```
<
XXX=Read      :    0x5E
XXX=Write     :    0x5F
>
```

<LOAD_SERVICE_BUFFER>

```
<
XXX=Read      :   Service_Buffer[1]=0x5E
                :   Service_Buffer[2]=Slot_Number
                :   Service_Buffer[3]=Index
                :   Service_Buffer[4]=Length
                :
XXX=Write     :   Service_Buffer[1]=0x5F
                :   Service_Buffer[2]=Slot_Number
                :   Service_Buffer[3]=Index
                :   Service_Buffer[4]=Length
                :   Service_Buffer[5..Length+5]=Data
>
```


<LOAD_ALARM_BUFFER>

```

<
    Alarm_Buffer[1]=0x5C
    Alarm_Buffer[2]=Slot_Number
    Alarm_Buffer[3]=Alarm_Type
    Alarm_Buffer[4]=Seq_Nr*8
>

```

IS_VALID_SERVICE

```

(
    Length    ≤ Max_Data_Length
)

```

IS_VALID_SERVICE_RES_PDU

```

(
    L_sdu[1]      = Service_Buffer[1] &&
    L_sdu.len      ≥ 4 &&
    (L_sdu[1] <> 0x5E || L_sdu.len ≥ L_sdu[4]+4) &&
    Max_Data_Length ≥ L_sdu[4]
)

```

IS_VALID_SERVICE_NRS_PDU

```

(
    L_sdu[1] = (Service_Buffer[1]+0x80) &&
    L_sdu.len = 4
)

```

IS_ALARM_ACK_PDU

```

(
    L_sdu[2..4] = Alarm_Buffer[2..4] &&
    (L_sdu[1] AND 0x7F = (Service_Buffer[1]) &&
    L_sdu.len = 4
)

```

<VALID_SERVICE>

```

<
    Service_Header=0x5E : MSAC1M_Read
    Service_Header=0x5F : MSAC1M_Write
>

```

<VALID_SERVICE_RES_PARAM>

```

<
    Service_Header=0x5E : Length=L_sdu[4],
                        Data=L_sdu[5..Length+4]
    Service_Header=0x5F : Length=L_sdu[4]
>

```

<VALID_SERVICE_NRS_PARAM>

```

<
    Error Decode=L_sdu[2],
    Error_Code_1=L_sdu[3],
    Error_Code_2=L_sdu[4],
>

```

REJECT_VALID_SERVICE

```
(
  IF (Stop_Pending=True) THEN
    (MSAC1M_Reject.ind(Rem_Add, Status=REJ_SE))
  ELSE IF (Go_Abort=True) THEN
    (MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT))
  ELSE IF (Src <> 0) THEN
    (MSAC1M_Reject.ind(Rem_Add, Status=REJ_PS))
  ELSE IF (Length<=Max_Data_Length) THEN
    (MSAC1M_Reject.ind(Rem_Add, Status=REJ_LE))
  ELSE
    (MSAC1M_Reject.ind(Rem_Add, Status=REJ_IV))
)
```

CONTINUE_VALID_SERVICE

```
(
  IF (Src <> 0 && Alarm_Dsap=51) THEN
    (
      Service_Header = Service_Buffer[1]
      DMPMM1_DATA_REPLY.req(SSAP=51, DSAP=51, Rem_Add,
        L_sdu=Service_Buffer, Serv_class=Low)
    )
  )
```

CONTINUE_ALARM_ACK

```
(
  IF (Arc <> 0 && Alarm_Dsap=51) THEN
    (
      Service_Header = Alarm_Buffer[1]
      DMPMM1_DATA_REPLY.req(SSAP=51, DSAP=51, Rem_Add,
        L_sdu=Alarm_Buffer, Serv_class=Low)
    )
  )
```

IS_INVALID_SERVICE_RES

```
(
  (L_status = DL &&
    (L_sdu[1]<> Service_Header OR L_sdu.len<4) &&
    (L_sdu[1]<> (Service_Header+0x80) OR L_sdu.len<> 4)
  ) OR
  L_status = DS/UE/RS/RDL/NA/RR/LR/DH/RDH
)
```

IS_INVALID_ALARM_ACK_RES

```
(
  (L_status = DL &&
    (L_sdu[1..4]<> Alarm_Buffer[1..4] OR L_sdu.len<> 4)
  ) OR
  L_status = DS/UE/RS/RDL/NA/RR/LR/DH/RDH
)
```

CHECK_STOP_PENDING

```
(
  IF Stop_Pending = TRUE THEN
    MSAC1M_Stop.cnf(Rem_Add)
  ELSE
    MSAC1M_Abort.ind(Rem_Add)
)
```

6.6.7.3 MSAC1M State Table

Table 420 - MSAC1M State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MSAC1M_MInit_MSAC1.req(Rem_Add) =>MSAC1M_MInit_MSAC1.cnf(Rem_Add)	CLOSED
2	CLOSED	MSAC1M_Start.req(Rem_Add) =>Alarm_Buffer := empty Service_Buffer := empty Stop_Pending := False Go_Abort := False Src := 0 Arc := 0 Service_Header := 0 Max_Data_Length := Max_Channel_Data_Length IF (Extra_Alarm_Sap:=0) THEN (Alarm_Dsap:=51) ELSE (Alarm_Dsap:=50) MSAC1M_Start.cnf(Rem_Add)	OPEN
3	CLOSED	MSAC1M_Stop.req(Rem_Add) =>MSAC1M_Stop.cnf(Rem_Add)	CLOSED
4	CLOSED	MSAC1M_XXX.req(Rem_Add, <REQPARAM>) =>MSAC1M_Reject.ind(Rem_Add, Status:=REJ_SE)	CLOSED
5	CLOSED	MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) =>ignore	CLOSED
6	CLOSED	DMPMM1_DATA_REPLY.cnf(SSAP, DSAP, Rem_Add, L_sdu, Serv_class, L_status) =>MSAC1M_Fault.ind(Rem_Add)	POWER-ON
7	OPEN	MSAC1M_Start.req(Rem_Add) =>MSAC1M_Fault.ind(Rem_Add)	POWER-ON
8	OPEN	MSAC1M_Stop.req(Rem_Add) /Arc = 0 && Src = 0 =>MSAC1M_Stop.cnf(Rem_Add)	CLOSED
9	OPEN	MSAC1M_Stop.req(Rem_Add) /Arc <> 0 Src <> 0 =>Stop_Pending:=True	OPEN
10	OPEN	MSAC1M_XXX.req(Rem_Add, <REQPARAM>) /Src = 0 && Stop_Pending = False && Go_Abort = False && IS_VALID_SERVICE_REQ && (Arc = 0 Alarm_Dsap <> 51) =>Service_Header := <VALID_SERVICE_FUNCTION_NUM> Src := Src+1 <LOAD_SERVICE_BUFFER> DMPMM1_DATA_REPLY.req(SSAP:=51, DSAP:=51, Rem_Add, L_sdu:=Service_Buffer, Serv_class:=Low)	OPEN
11	OPEN	MSAC1M_XXX.req(Rem_Add, <REQPARAM>) /Src = 0 && Stop_Pending = False && Go_Abort = False && IS_VALID_SERVICE_REQ && Arc <> 0 && Alarm_Dsap = 51 =>Src := Src+1 <LOAD_SERVICE_BUFFER>	OPEN
12	OPEN	MSAC1M_XXX.req(Rem_Add, <REQPARAM>) /Src <> 0 Stop_Pending = True Go_Abort = True NOT (IS_VALID_SERVICE_REQ) =>REJECT_VALID_SERVICE	OPEN
13	OPEN	MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Arc = 0 && Stop_Pending = False && Go_Abort = False && Alarm_Dsap <> 51 =>Arc := Arc+1 <LOAD_ALARM_BUFFER> DMPMM1_DATA_REPLY.req(SSAP:=51, DSAP:=Alarm_Dsap, Rem_Add, L_sdu:=Alarm_Buffer, Serv_class:=Low)	OPEN
14	OPEN	MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Arc = 0 && Stop_Pending = False && Go_Abort = False && Src = 0 && Alarm_Dsap = 51 =>Arc := Arc+1 DMPMM1_DATA_REPLY.req(SSAP:=51, DSAP:=Alarm_Dsap, Rem_Add, L_sdu:=Alarm_Buffer, Serv_class:=Low)	OPEN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
15	OPEN	MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Arc = 0 && Stop_Pending = False && Go_Abort = False && Src <> 0 && Alarm_Dsap = 51 =>Arc := Arc+1 <LOAD_ALARM_BUFFER>	OPEN
16	OPEN	MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Arc=0 && (Stop_Pending = True Go_Abort = True) =>ignore	OPEN
17	OPEN	MSAC1M_Alarm_Ack.req(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr) /Arc <> 0 =>DMPMM1_Fault.ind(Rem_Add, Status:=MSAC1_Fault)	POWER- ON
18	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_RES_PDU && Src <> 0 && L_sdu[1] = Service_Header && Stop_Pending = False && Go_Abort = False =>Src := 0 CONTINUE_ALARM_ACK MSAC1_<VALID_SERVICE>.cnf(+)(Rem_Add, <VALID_SERVICE_RES_PARAM>)	OPEN
19	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_RES_PDU && Src <> 0 && L_sdu[1] = Service_Header && Stop_Pending = True && (Arc=0 Alarm_Dsap=51) =>Src := 0 Arc := 0 MSAC1_<VALID_SERVICE>.cnf(+)(Rem_Add, <VALID_SERVICE_RES_PARAM>) MSAC1M_Stop.cnf(Rem_Add)	CLOSED
20	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_RES_PDU && Src <> 0 && L_sdu[1] = Service_Header && Stop_Pending = True && Arc <> 0 && Alarm_Dsap <> 51 =>Src := 0 MSAC1_<VALID_SERVICE>.cnf(+)(Rem_Add, <VALID_SERVICE_RES_PARAM>)	OPEN
21	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_RES_PDU && Src <> 0 && L_sdu[1] = Service_Header && Stop_Pending = False && Go_Abort = True =>Src := 0 MSAC1_<VALID_SERVICE>.cnf(+)(Rem_Add, <VALID_SERVICE_RES_PARAM>) MSAC1M_Abort.ind(Rem_Add)	CLOSED
22	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_NRS_PDU && Src <> 0 && L_sdu[1] = (Service_Header+0x80) && Stop_Pending = False && Go_Abort = False =>Src := 0 CONTINUE_ALARM_ACK MSAC1_<VALID_SERVICE>.cnf(-)(Rem_Add, <VALID_SERVICE_NRS_PARAM>)	OPEN
23	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_NRS_PDU && Src <> 0 && L_sdu[1] = (Service_Header+0x80) && Stop_Pending = True && (Arc=0 Alarm_Dsap=51) =>Src := 0 Arc := 0 MSAC1_<VALID_SERVICE>.cnf(-)(Rem_Add, <VALID_SERVICE_NRS_PARAM>) MSAC1M_Stop.cnf(Rem_Add)	CLOSED
24	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_NRS_PDU && Src <> 0 && L_sdu[1] = (Service_Header+0x80) && Stop_Pending = True && Arc <> 0 && Alarm_Dsap <> 51 =>Src := 0 MSAC1_<VALID_SERVICE>.cnf(-)(Rem_Add, <VALID_SERVICE_NRS_PARAM>)	OPEN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
25	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_SERVICE_NRS_PDU && Src <> 0 && L_sdu[1] = (Service_Header+0x80) && Stop_Pending = False && Go_Abort = True =>Src := 0 MSAC1M_<VALID_SERVICE>.cnf(-)(Rem_Add, <VALID_SERVICE_NRS_PARAM>) MSAC1M_Abort.ind(Rem_Add)	CLOSED
26	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_ALARM_ACK_PDU && Arc <> 0 && L_sdu[1]=Service_Buffer[1] && (Alarm_Dsap<>51 L_sdu[1]=Service_Header) && Stop_Pending = False && Go_Abort = False =>Arc := 0 CONTINUE_VALID_SERVICE MSAC1S_Alarm_Ack.cnf(+)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr)	OPEN
27	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_ALARM_ACK_PDU && Arc <> 0 && L_sdu[1]=(Service_Buffer[1] OR 0x80) && (Alarm_Dsap<>51 L_sdu[1]=Service_Header) && Stop_Pending = False && Go_Abort = False =>Arc := 0 CONTINUE_VALID_SERVICE MSAC1S_Alarm_Ack.cnf(-)(Rem_Add, Slot_Number, Alarm_Type, Seq_Nr)	OPEN
28	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_ALARM_ACK_RES_PDU && Arc <> 0 && (Alarm_Dsap<>51 L_sdu[1]=Service_Header) && Stop_Pending = True && Src = 0 =>Arc := 0 MSAC1M_Stop.cnf(Rem_Add)	CLOSED
29	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_ALARM_ACK_PDU && Arc <> 0 && Alarm_Dsap = 51 && L_sdu[1] = Service_Header && Stop_Pending = True && Src <> 0 =>Arc := 0 Src := 0 MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT) MSAC1M_Stop.cnf(Rem_Add)	CLOSED
30	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_ALARM_ACK_PDU && Arc <> 0 && Alarm_Dsap <> 51 && Stop_Pending = True && Src <> 0 =>Arc := 0	OPEN
31	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = DL && IS_VALID_ALARM_ACK_PDU && Arc <> 0 && (Alarm_Dsap<>51 L_sdu[1]=Service_Header) && Stop_Pending = False && Go_Abort = True =>Arc := 0 MSAC1M_Abort.ind(Rem_Add)	CLOSED
32	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Src <> 0 && Service_Header <> Alarm_Ack && Stop_Pending = False && Go_Abort = False =>DMPMM1_DATA_REPLY.req(SSAP=51, DSAP, Rem_Add, L_sdu.len=0, Serv_class=Low)	OPEN
33	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Src <> 0 && Service_Header <> Alarm_Ack && Stop_Pending = True && (Arc=0 Alarm_Dsap=51) =>Src := 0 Arc := 0 MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT) MSAC1M_Stop.cnf(Rem_Add)	CLOSED

#	Current State	Event /Condition =>Actions taken&&Output	Next State
34	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Src <> 0 && Service_Header <> Alarm_Ack && Stop_Pending = True && Arc <> 0 && Alarm_Dsap <> 51 =>Src := 0 MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT)	OPEN
35	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Src <> 0 && Service_Header <> Alarm_Ack && Stop_Pending = False && Go_Abort = True =>Src := 0 MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT) MSAC1M_Abort.ind(Rem_Add)	CLOSED
36	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Arc <> 0 && (Service_Header=Alarm_Ack Alarm_Dsap<>51) && Stop_Pending = False && Go_Abort = False =>DMPMM1_DATA_REPLY.req(SSAP=51, DSAP, Rem_Add, L_sdu.len=0, Serv_class=Low)	OPEN
37	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Arc <> 0 && (Service_Header=Alarm_Ack Alarm_Dsap<>51) && Stop_Pending = True && Src = 0 =>Arc := 0 MSAC1M_Stop.cnf(Rem_Add)	CLOSED
38	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Arc <> 0 && Service_Header = Alarm_Ack && Alarm_Dsap = 51 && Stop_Pending = True && Src <> 0 =>Arc := 0 Src := 0 MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT) MSAC1M_Stop.cnf(Rem_Add)	CLOSED
39	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Arc <> 0 && Alarm_Dsap <> 51 && Stop_Pending = True && Src <> 0 =>Arc := 0	OPEN
40	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=Alarm_Dsap, Rem_Add, L_sdu, Serv_class=LOW, L_status) /L_status = NR && Arc <> 0 && (Service_Header=Alarm_Ack Alarm_Dsap<>51) && Stop_Pending = False && Go_Abort = True =>Arc := 0 MSAC1M_Abort.ind(Rem_Add)	CLOSED
41	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /IS_INVALID_SERVICE_RES && Src <> 0 && Service_Header <> Alarm_Ack && (Arc=0 Alarm_Dsap=51) =>Src := 0 Arc := 0 CHECK_STOP_PENDING MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT)	CLOSED
42	OPEN	DMPMM1_DATA_REPLY.cnf(SSAP=51, DSAP=51, Rem_Add, L_sdu, Serv_class=LOW, L_status) /IS_INVALID_SERVICE_RES && Src <> 0 && Service_Header <> Alarm_Ack && Arc <> 0 && Alarm_Dsap <> 51 =>Src := 0 IF Stop_Pending := False THEN Go_Abort := True MSAC1M_Reject.ind(Rem_Add, Status=REJ_ABORT)	OPEN

6.6.8 MMAC1

6.6.8.1 Primitive Definitions

6.6.8.1.1 Primitives Exchanged between FSPMM1 and MMAC1

Table 421 - Primitives issued by FSPMM1 to MMAC1

Primitive Name	Source	Associated Parameters	Functions
Sinit MM.req	FSPMM1	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMM1	(none)	
Abort.req	FSPMM1	(none)	
Get Master Diag.rsp(+)	FSPMM1	Diagnosis Data	
Get Master Diag.rsp(-)	FSPMM1	Status	
Start Seq.rsp(+)	FSPMM1	Max Len Data Unit	
Start Seq.rsp(-)	FSPMM1	Status	
Download.rsp(+)	FSPMM1	(none)	
Download.rsp(-)	FSPMM1	Status	
Upload.rsp(+)	FSPMM1	Data	
Upload.rsp(-)	FSPMM1	Status	
End Seq.rsp(+)	FSPMM1	(none)	
End Seq.rsp(-)	FSPMM1	Status	
Act Param.rsp(+)	FSPMM1	(none)	
Act Param.rsp(-)	FSPMM1	Status	

Table 422 - Primitives issued by MMAC1 to FSPMM1

Primitive Name	Source	Associated Parameters	Functions
Sinit MM.cnf	MMAC1	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	MMAC1	(none)	
Abort.ind	MMAC1	(none)	
Fault.ind	MMAC1	(none)	
Get Master Diag.ind	MMAC1	MDiag Identifier	
Start Seq.ind	MMAC1	Area Code, Timeout	
Download.ind	MMAC1	Area Code, Add Offset, Data	
Upload.ind	MMAC1	Area Code, Add Offset, Data Len	
End Seq.ind	MMAC1	(none)	
Act Param.ind	MMAC1	Area Code, Activate	
Act Para Brct.ind	MMAC1	Area Code	

6.6.8.1.2 Parameters of MMAC1 Primitives

The parameters used with the primitives exchanged between the FSPMM1 and the MMAC1 are described in FAL Type 3 Service Definition IEC 61158-5.

6.6.8.2 State Machine Description

The following MMAC1 state machine for Master-Master communication describes the responder side. Normally, this means a DP-interface of an automation device such as a Programmable Logic Controller.

After Reset, the MMAC1 changes to the "POWER-ON" state and waits for initialisation.

The state machine remains in the state "WAIT-REQ" as long as it receives a valid request from Layer 2. After issuing this request to the User, the state machine remains in the "WAIT-RES" state until the User offers the response. After storing the response, data collection is monitored. If a service-sequence is executed, MMAC1 is waiting for the next service request in the state "WAIT-SEQ-REQ".

At any time, maximally one request shall be in execution.

a) Local variables of the MMAC1

T1

(Unsigned16)

Time in which a Master (Class 2) shall receive at latest a response from the Master (Class 1).

T2

(Unsigned16)

Time that can elapse during the execution of a service-sequence between response and the following request.

Client_Add

(Unsigned8)

Address of the DP-Master (Class 2) just communicating with.

Service-Header

(Octet-String of the length 4)

Intermediate memory for the first four octets of the last request.

Sequence_Mode

(Boolean)

Indicates that a service-sequence is just in execution. During a service-sequence the DP-Master (Class 1) is reserved for the respective DP-Master (Class 2).

6.6.8.3 MMAC1 State Table

Table 423 - MMAC1 State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MMAC1_Minit_MM.req =>MMAC1_Minit_MM.cnf	WAIT-REQ
2	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=0 =>ignore	WAIT-REQ
3	WAIT-REQ	MMAC1_XXX.rsp =>MMAC1_Reject.ind(Reason:=REJ_PS)	WAIT-REQ
4	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=2 && L_sdu[1]=0x41 =>Service_Header:=L_sdu[1-2] Client_Add:=Loc_add MMAC1_Get_Master_Diag.ind(Identifier:=L_sdu[2]) DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	WAIT-PROT

#	Current State	Event /Condition =>Actions taken&&Output	Next State
5	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=4 && L_sdu[1]=0x42 =>Sequence_Mode:=True Service_Header:=L_sdu[1-4] T2:=L_sdu[3-4] Client_Add:=Loc_add MMAC1_Start_Seq.ind(Req_Add:=Loc_add, Area_Code:=L_sdu[2], Timeout:=L_sdu[3-4]) DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	WAIT-PROT
6	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len>=5 && L_sdu[1]=0x43 =>Service_Header:=L_sdu[1-4] Client_Add:=Loc_add MMAC1_Download.ind(Req_Add:=Loc_add, Area_Code:=L_sdu[2], Address_Offset:=L_sdu[3-4], Data:=L_sdu[5-L_sdu.len]) DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	WAIT-PROT
7	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=5 && L_sdu[1]=0x44 =>Service_Header:=L_sdu[1-4] Client_Add:=Loc_add MMAC1_Upload.ind(Req_Add:=Loc_add, Area_Code:=L_sdu[2], Address_Offset:=L_sdu[3-4], Data_Length:=L_sdu[5]) DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	WAIT-PROT
8	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=1 && L_sdu[1]=0x45 =>Client_Add:=Loc_add DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	END-SEQ-PROT
9	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=3 && L_sdu[1]=0x47 =>Service_Header:=L_sdu[1-3] Client_Add:=Loc_add MMAC1_Act_Param.ind(Area_Code:=L_sdu[2], Activate:=L_sdu[3]) DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	WAIT-PROT
10	WAIT-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /invalid MMAC-REQ-PDU =>Client_Add:=Loc_add DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=Loc_add, Indication_Mode:=Unchanged)	ERR-PROT
11	WAIT-REQ	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=2 && L_sdu[1]=0x46 =>MMAC1_Act_Para_Brct.ind(Area_Code:=L_sdu[2])	WAIT-REQ
12	WAIT-REQ	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len<>2 L_sdu[1]<>0x46 =>ignore	WAIT-REQ
13	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=0 =>ignore	WAIT-REQ
14	WAIT-SEQ-REQ	MMAC1_xxx.rsp =>Sequence_Mode:=False Stop T2 DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=All, Indication_Mode:=Unchanged)	WAIT-UNPROT
15	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=2 && L_sdu[1]=0x41 =>Service_Header:=L_sdu[1-2] Stop T2 MMAC1_Get_Master_Diag.ind(Identifier:=L_sdu[2])	WAIT-RES
16	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=4 && L_sdu[1]=0x42 =>Sequence_Mode:=False Stop T2 DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu[1]:=0xC9, Serv_class:=Low, Transmit:=Single) {SE error code}	WAIT-UPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
17	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len>=5 && L_sdu[1]=0x43 =>Service_Header:=L_sdu[1-4] Stop T2 MMAC1_Download.ind(Req_Add:=Loc_add, Area_Code:=L_sdu[2], Address_Offset:=L_sdu[3-4], Data:=L_sdu[5-L_sdu.len])	WAIT-RES
18	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=5 && L_sdu[1]=0x44 =>Service_Header:=L_sdu[1-4] Stop T2 MMAC1_Upload.ind(Req_Add:=Loc_add, Area_Code:=L_sdu[2], Address_Offset:=L_sdu[3-4], Data_Length:=L_sdu[5])	WAIT-RES
19	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=1 && L_sdu[1]=0x45 =>Sequence_Mode:=False Service_Header:=L_sdu[1] Stop T2 MMAC1_End_Seq.ind(Req_Add:=Loc_add)	WAIT-RES
20	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /L_sdu.len=3 && L_sdu[1]=0x47 =>Service_Header:=L_sdu[1-3] Stop T2 MMAC1_Act_Param.ind(Area_Code:=L_sdu[2], Activate:=L_sdu[3])	WAIT-RES
21	WAIT-SEQ-REQ	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /invalid MMAC-REQ-PDU && Sequence_Mode=True =>Sequence_Mode:=False Stop T2 DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu[1]:=0xC1, Serv_class:=Low, Transmit:=Single) {FE error code}	WAIT-UPD
22	WAIT-SEQ-REQ	T2 expired =>Sequence_Mode:=False DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=All, Indication_Mode:=Unchanged)	WAIT-UNPROT
23	WAIT-SEQ-REQ	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class) /L_sdu.len=2 && Loc_add=Client_Add && L_sdu[1]=0x46 =>MMAC1_Act_Para_Brc.ind(Area_Code:=L_sdu[2])	WAIT-SEQ-REQ
24	WAIT-SEQ-REQ	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class) /L_sdu.len<>2 Loc_add<>Client_Add L_sdu[1]<>0x46 =>ignore	WAIT-SEQ-REQ
25	END-SEQ-PROT	DMPMM1_RSAP_ACTIVATE.cnf(SSAP=54, M_status) /M_status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu[1]:=0xC9, Serv_class:=Low, Transmit:=Single) {SE error code}	WAIT-UPD
26	ERR-PROT	DMPMM1_RSAP_ACTIVATE.cnf(SSAP=54, M_status) /M_status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu[1]:=0xC1, Serv_class:=Low, Transmit:=Single) {FE error code}	WAIT-UPD
27	WAIT-PROT	DMPMM1_RSAP_ACTIVATE.cnf(SSAP=54, M_status) /M_status=OK	WAIT-RES
28	WAIT-RES	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Sequence_Mode=False Client_Add<>Loc_add L_sdu.len=0 =>ignore	WAIT-RES
29	WAIT-RES	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Sequence_Mode=True && Client_Add=Loc_add && L_sdu.len>0 =>Sequence_Mode:=False ignore L_sdu	WAIT-RES
30	WAIT-RES	MMAC1_Get_Master_Diag.rsp(Status, Diagnostic_Data) /Service_Header[1]=0x41 && Status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu:=(Service_Header, Diagnostic_Data), Serv_class:=Low, Transmit:=Single)	WAIT-UPD
31	WAIT-RES	MMAC1_Start_Seq.rsp(Status, Max_Length_Data_Unit) /Service_Header[1]=0x42 && Status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu:=(Service_Header, Max_Length_Data_Unit), Serv_class:=Low, Transmit:=Single)	WAIT-UPD

#	Current State	Event /Condition =>Actions taken&&Output	Next State
32	WAIT-RES	MMAC1_Download.rsp(Status) /Service_Header[1]=0x43 && Status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu:=Service_Header, Serv_class:=Low,Transmit:=Single)	WAIT-UPD
33	WAIT-RES	MMAC1_Upload.rsp(Status, Data) /Service_Header[1]=0x44 && Status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu:=(Service_Header, Data), Serv_class:=Low,Transmit:=Single)	WAIT-UPD
34	WAIT-RES	MMAC1_End_Seq.rsp(Status) /Service_Header[1]=0x45 && Status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu:=Service_Header, Serv_class:=Low,Transmit:=Single)	WAIT-UPD
35	WAIT-RES	MMAC1_Act_Param.rsp(Status) /Service_Header[1]=0x47 && Status=OK =>DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu:=Service_Header, Serv_class:=Low,Transmit:=Single)	WAIT-UPD
36	WAIT-RES	MMAC1_xxx.rsp(Status) /Service_Header[1]=Service_Code && Status <> OK =>Start T1 DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu[1]:=Status, Serv_class:=Low, Transmit:=Single)	WAIT-UPD
37	WAIT-RES	MMAC1_XXX.rsp(Status) /Service_Header[1]<>Service_Code =>Sequence_Mode:=False DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu[1]:=0xC6, Serv_class:=Low, Transmit:=Single) {RE}	WAIT-UPD
38	WAIT-UPD	DMPMM1_REPLY_UPDATE.cnf(SSAP=54, Serv_class, L_status) /Status=OK =>Start T1	WAIT-IND
39	WAIT-UPD	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Loc_add<>Client_Add && Update_status=NO =>ignore	WAIT-UPD
40	WAIT-UPD	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Loc_add=Client_Add && L_sdu.len>0 && Update_status=NO =>Sequence_Mode:=False ignore L_sdu	WAIT-UPD
41	WAIT-UPD	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Update_status=LO/Hi =>MMAC1_Fault.ind	POWER-ON
42	WAIT-UPD	DMPMM1_REPLY_UPDATE.cnf(SSAP=54) /Status=NO/IV =>MMAC1_Fault.ind	POWER-ON
43	WAIT-UPD	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /(Update_status=LO/Hi) (Update_status=NO && L_sdu.len=0) =>MMAC1_Fault.ind	POWER-ON
44	WAIT-UPD	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class) /L_sdu.len=2 && L_sdu[1]=0x46 && (Sequence_Mode=False Loc_add=Client_Add) =>MMAC1_Act_Para_Brct.ind(Area_Code:=L_sdu[2])	WAIT-UPD
45	WAIT-UPD	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class) /L_sdu.len<>2 L_sdu[1]<>0x46 Sequence_Mode=True Loc_add<>Client_Add =>ignore	WAIT-UPD
46	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Loc_add<>Client_Add && Update_status=NO =>ignore	WAIT-IND
47	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Update_status=LO/Hi && Loc_add<>Client_Add =>stop T1 MMAC1_Fault.ind	POWER-ON
48	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /(Update_status=LO && Loc_add<>Client_Add) (Update_status=NO && L_sdu.len=0) (Update_status=Hi) =>MMAC1_Fault.ind	POWER-ON

#	Current State	Event /Condition =>Actions taken&&Output	Next State
49	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Loc_add=Client_Add && L_sdu.len>0 && Update_status=NO =>Sequence_Mode = False ignore L_sdu	WAIT-IND
50	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Update_status=LO && Sequence_Mode=False && Loc_add = Client_Add =>ignore L_sdu, if existent Stop T1 DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=All, Indication_Mode:=Unchanged)	WAIT-UNPROT
51	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Update_status=LO && Sequence_Mode=True && L_sdu.len=0 && Loc_add = Client_Add =>Stop T1 Start T2	WAIT-SEQ-REQ
52	WAIT-IND	DMPMM1_DATA_REPLY.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class, Update_status) /Update_status=LO && Sequence_Mode=True && L_sdu.len>0 && Loc_add = Client_Add =>Sequence_Mode:=False ignore L_sdu, if existent STOP T1 DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=All, Indication_Mode:=Unchanged)	WAIT-UNPROT
53	WAIT-IND	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class) /L_sdu.len=2 && L_sdu[1]=0x46 && (Sequence_Mode=False Loc_add=Client_Add) =>MMAC1_Act_Para_Brct.ind(Area_Code:=L_sdu[2])	WAIT-IND
54	WAIT-IND	DMPMM1_DATA.ind(SSAP,DSAP=54,Loc_add,Rem_add,L_sdu,Serv_class) /L_sdu.len<>2 L_sdu[1]<>0x46 (Sequence_Mode=True && Loc_add<>Client_Add) =>ignore	WAIT-IND
55	WAIT-IND	T1 expired =>Sequence_Mode:=False DMPMM1_REPLY_UPDATE.req(SSAP:=54, L_sdu.len=0, Serv_class:=Low, Transmit:=Single)	WAIT-DEL
56	WAIT-DEL	DMPMM1_REPLY_UPDATE.cnf(SSAP=54) /L_status=OK =>DMPMM1_RSAP_ACTIVATE.req(SSAP:=54, Access:=All, Indication_Mode:=Unchanged)	WAIT-UNPROT
57	ANY-STATE	MMAC1_Reset.req =>Stop T1 MMAC1_Reset.cnf	POWER-ON

6.6.9 MSAC2M

6.6.9.1 Primitive Definitions

6.6.9.1.1 Primitives Exchanged between FSPMM2 and MSAC2M

Table 424 - Primitives issued by FSPMM2 to MSAC2M

Primitive Name	Source	Associated Parameters	Functions
MInit MS2.req	FSPMM2	C_Ref	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMM2	(none)	
Abort.req	FSPMM2	C_Ref	
Initiate.req	FSPMM2	C_Ref, Send Timeout, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
Read.req	FSPMM2	C_Ref, Slot Number, Index, Length	

Primitive Name	Source	Associated Parameters	Functions
Write.req	FSPMM2	C_Ref, Slot Number, Index, Length, Data	
Data Transport.req	FSPMM2	C_Ref, Slot Number, Index, Length, Data	

Table 425 - Primitives issued by MSAC2M to FSPMM2

Primitive Name	Source	Associated Parameters	Functions
MInit MS2.cnf	MSAC2M	C_Ref	
Reset.cnf	MSAC2M	(none)	
Initiate.cnf(+)	MSAC2M	C_Ref, Max Len Data Unit, Features Supported, Profile Features Supported, Profile Ident Number, Add Addr Param	
Initiate.cnf(-)	MSAC2M	C_Ref, Error Decode, Error Code 1 Error Code 2	
Read.cnf(+)	MSAC2M	C_Ref, Length, Data	
Read.cnf(-)	MSAC2M	C_Ref, Error Decode, Error Code 1 Error Code 2	
Write.cnf(+)	MSAC2M	C_Ref, Length	
Write.cnf(-)	MSAC2M	C_Ref, Error Decode, Error Code 1 Error Code 2	
Data Transport.cnf(+)	MSAC2M	C_Ref, Length, Data	
Data Transport.cnf(-)	MSAC2M	C_Ref, Error Decode, Error Code 1 Error Code 2	
Reject.ind	MSAC2M	C_Ref, Reason Code	
Abort.ind	MSAC2M	C_Ref, Locally_Generated, Subnet, Instance, Reason_Code, Abort_Detail	
Fault.ind	MSAC2M	C_Ref	

6.6.9.1.2 Parameters of MSAC2M Primitives

The parameters used with the primitives exchanged between the FSPMM2 and the MSAC2M are described in FAL Type 3 Service Definition IEC 61158-5.

Additionally used parameters are:

Table 426 – Parameters Used with Primitives Exchanged with MSAC2M

Parameter Name	Description
C_Ref	Local identifier of a MSAC2 state machine

6.6.9.2 State Machine Description

On POWER-ON each MSAC2M State Machine waits for initialisation.

In the state CLOSED the Initiate service called from the AP-Context is expected and sent to the DP-Slave. In the state START-POLL-RES the Master is waiting for the positive Initiate.cnf and makes the transition into the state OPEN.

In the OPEN-state the services Read, Write and Data_Transport are allowed. The State Machine controls that only one request from the User is processed simultaneously, otherwise the connection is aborted.

The connection can be aborted by the User of the Master or by the User of the Slave.

The connection is aborted automatically if a protocol error is detected.

The State Machine needs some local variables which are described in clause a). Furthermore the machine uses macros which are listed in clause b).

a) Local variables of the MSAC2M

Client_SAP

(Unsigned8)

The SAP used for the connection.

Server_SAP

(Unsigned8)

The SAP used for the connection at the server side.

Server_Add

(Unsigned8)

The DL address (0-126) of the remote server.

S-Timer

(Unsigned16)

This timer monitors the sending of service request PDUs on the connection if no service is outstanding. The S-Timer expires if no service request PDU has been sent during the S-Timer interval. In this case an Idle-REQ-PDU will be sent.

R-Timer

(Unsigned16)

This Timer monitors the liveliness of the connection if there is an outstanding service response. The R-Timer expires if no service response or IDLE-PDU during the R-Timer interval has been received. In this case the connection will be aborted. The R-Timer interval is 3 times the S-Timer interval.

Service-Header

Local buffer for the Function_Num of the last transmitted request.

Go_Abort

(Boolean)

Local flag that indicates an abort reaction after receiving the remote response.

Service_Buffer

(Octet-String[255])

Local buffer that stores one complete service during an outstanding Idle-RES-PDU.

b) Macros

;XXX handles Data_Transport, Read and Write Services.

<REQPARAM>

```
<
XXX=Read:      Slot_Number, Index, Length
XXX=Write:     Slot_Number, Index, Length, Data
XXX=Data_Transport: Slot_Number, Index, Length, Data
>
```

<VALID_SERVICE_FUNCTION_NUM>

```
<
XXX=Read      :      0x5E
XXX=Write     :      0x5F
XXX=Data_Transport :    0x51
>
```

<LOAD_SERVICE_BUFFER>

```
<
XXX=Read :      Service_Buffer[1]=0x5E
              Service_Buffer[2]=Slot_Number
              Service_Buffer[3]=Index
              Service_Buffer[4]=Length

XXX=Write :      Service_Buffer[1]=0x5F
              Service_Buffer[2]=Slot_Number
              Service_Buffer[3]=Index
              Service_Buffer[4]=Length
              Service_Buffer[5..Length+4]=Data

XXX=Data_Transport: Service_Buffer[1]=0x51
              Service_Buffer[2]=Slot_Number
              Service_Buffer[3]=Index
              Service_Buffer[4]=Length
              Service_Buffer[5..Length+4]=Data
>
```

VALID_SERVICE

```
(
Length ≤ Max_Data_Length
)
```

VALID_SERVICE_RES_PDU

```
(
L_sdu[1] = Service_Buffer[1] &&
L_sdu.len ≥ 4 &&
(L_sdu[1] <> 0x5E || L_sdu.len ≥ L_sdu[4]+4) &&
Max_Data_Length ≥ L_sdu[4]
)
```

VALID_SERVICE_NRS_PDU

```
(
L_sdu[1] = (Service_Buffer[1]+0x80) &&
L_sdu.len = 4
)
```

<VALID_SERVICE>

```
<
Service_Header=0x5E :      Read
Service_Header=0x5F :      Write
Service_Header=0x51 :      Data_Transport
>
```

<VALID_SERVICE_RES_PARAM>

```
<
Service_Header=0x5E :      Length=L_sdu[4],
                           Data=L_sdu[5..Length+4]
Service_Header=0x5F :      Length=L_sdu[4]
Service_Header=0x51 :      Length=L_sdu[4],
                           Data=L_sdu[5..Length+4]
>
```

<VALID_SERVICE_NRS_PARAM>

```
<
Error_Decode=L_sdu[2],
Error_Code_1=L_sdu[3],
Error_Code_2=L_sdu[4],
>
```

VALID_SERVICE_REQ_PDU

```
(
  (Read-REQ-PDU) OR
  (Write-REQ-PDU) OR
  (Data_Transport-REQ-PDU)
)
```

VALID_SERVICE_RES_PDU

```
(
  (Read-RES-PDU) OR
  (Write-RES-PDU) OR
  (Data_Transport-RES-PDU)
)
```

VALID_SERVICE_NRS_PDU

```
(
  (Read-NRS-PDU) OR
  (Write-NRS-PDU) OR
  (Data_Transport-NRS-PDU)
)
```

STORE_ABORT_PARAMETER

```
(
  Stored-Subnet=Subnet
  Stored-Instance=Instance
  Stored-Reason_Code=Reason_Code
)
```

6.6.9.3 MSAC2M State Table

Table 427 - MSAC2M State Table

#	Current State	Event /Condition =>Actions taken Output	Next State
1	POWER-ON	MSAC2M_Minit_MS2.req(C_Ref) =>Stored_RM_SAP := 49 Stored_C_Ref := C_Ref Stored_Max_Local_Data_Length := Max_Local_Data_Length Client_SAP := 50 MSAC2M_Minit_MS2.cnf(C_Ref)	CLOSED
2	CLOSED	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>Server_Add := Rem_Add S-Timer := Send_Timeout R-Timer := 3*Send_Timeout Go_Abort := FALSE Service_Buffer := empty store Initiate-REQ-PDU Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=STORED_RM_SAP, Rem_Add:=Server_Add, L_sdu:=Initiate-REQ-PDU, Serv_class:=Low)	START-POLL-RMREQ
3	CLOSED	MSAC2M_Abort.req(C_Ref:=Stored_C_Ref, Subnet, Instance, Reason_Code) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
4	CLOSED	MSAC2M_XXX.req(C_Ref:=Stored_C_Ref, <REQPARAM>) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
5	CLOSED	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = IV/LS =>ignore	CLOSED
6	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = RM-REQ-PDU && S-Timer >= Send_Timeout && Go_Abort = FALSE =>Stored_Server_SAP := Server_SAP Stop S-Timer Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	START-POLL-RES
7	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = RM-REQ-PDU && Go_Abort = TRUE =>Stored_Server_SAP := Server_SAP Go_Abort := TRUE STORE_ABORT_PARAMETER Stop S-Timer Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	START-POLL-RES
8	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = RM-REQ-PDU && S-Timer < Send_Timeout =>Stored_Server_SAP := Server_SAP Go_Abort := TRUE STORE_ABORT_PARAMETER Stop S-Timer Start R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_STO, Additional_Detail:=Send_Timeout) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	START-POLL-RES

#	Current State	Event /Condition =>Actions taken Output	Next State
9	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu <> RM-REQ-PDU =>Stop S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_FE) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
10	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = RR && Go_Abort = FALSE =>DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=STORED_RM_SAP, Rem_Add:=Server_Add, L_sdu:=stored Initiate-REQ-PDU, Serv_class:=Low)	START-POLL-RMREQ
11	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = RR && Go_Abort = TRUE =>Stop S-Timer MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
12	START-POLL-RMREQ	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_RM_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = RS/NR/RDL/RDH/LR/NA/DH/DS/UE =>ignore L_sdu if present Stop S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=FALSE, Instance:=FDL, Reason_Code:=L_Status) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
13	START-POLL-RMREQ	MSAC2M_Abort.req(C_Ref:=Stored_C_Ref, Subnet, Instance, Reason_Code) =>Go_Abort := TRUE STORE_ABORT_PARAMETER	START-POLL-RMREQ
14	START-POLL-RMREQ	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>Go_Abort := TRUE STORE_ABORT_PARAMETER MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	START-POLL-RMREQ
15	START-POLL-RMREQ	MSAC2M_XXX.req(C_Ref:=Stored_C_Ref, <REQPARAM>) =>Go_Abort := TRUE STORE_ABORT_PARAMETER MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	START-POLL-RMREQ
16	START-POLL-RMREQ	S-Timer expired =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_TO)	FINAL-WAIT-CON
17	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = RS/NR =>DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	START-POLL-RES
18	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Initiate-RES-PDU && Go_Abort = FALSE =>Stored_Max_Len_Data_Unit := Max_Len_Data_Unit Stop R-Timer Start S-Timer MSAC2M_Initiate.cnf(+, C_Ref:=Stored_C_Ref, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param)	OPEN
19	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Initiate-RES-PDU && Go_Abort = TRUE =>Stop R-Timer Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU with stored Parameter, Serv_class:=Low)	END-POLL-RES

#	Current State	Event /Condition =>Actions taken Output	Next State
20	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DS/UE =>Stop R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
21	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = RDL/NA/LR/DH/RDH/RR =>ignore L_sdu if present Stop R-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
22	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Abort-REQ-PDU =>Stop R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=FALSE, Subnet, Instance, Reason_Code) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
23	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = (Initiate-NRS-PDU) && Go_Abort = FALSE =>Stop R-Timer MSAC2M_Initiate.cnf(-, C_Ref:=Stored_C_Ref, Error_Decode, Error_Code_1, Error_Code_2) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
24	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = (Initiate-NRS-PDU) && Go_Abort = TRUE =>Stop R-Timer MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
25	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && NOT (L_sdu = Initiate-RES-PDU L_sdu = (Initiate-NRS-PDU) L_sdu = Abort-REQ-PDU L_sdu = Idle-REQ-PDU) =>ignore L_sdu Stop R-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_RE) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
26	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Idle-REQ-PDU && Go_Abort = FALSE =>Stop R-Timer Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Idle-RES-PDU, Serv_class:=Low)	START-POLL-RES
27	START-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Idle-REQ-PDU && Go_Abort = TRUE =>Stop R-Timer Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU with stored Parameter, Serv_class:=Low)	END-POLL-RES
28	START-POLL-RES	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>STORE_ABORT_PARAMETER Go_Abort := TRUE MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	START-POLL-RES

#	Current State	Event /Condition =>Actions taken Output	Next State
29	START-POLL-RES	MSAC2M_Abort.req(C_Ref=Stored_C_Ref, Subnet, Instance, Reason_Code) =>Go_Abort := TRUE STORE_ABORT_PARAMETER	START-POLL-RES
30	START-POLL-RES	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) =>Go_Abort := TRUE STORE_ABORT_PARAMETER MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	START-POLL-RES
31	START-POLL-RES	R-Timer expired =>STORE_ABORT_PARAMETER Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_TO)	END-WAIT-CON
32	OPEN	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) /Length <= (Stored_Max_Data_Length) && Length <= (Stored_Max_Local_Data_Length) =>Service_Header := Function_Num Stop S-Timer Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=VALID_SERVICE_REQ_PDU, Serv_class:=Low)	VALID-SERVICE-POLL-RES
33	OPEN	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) /Length > (Stored_Max_Data_Length) Length > (Stored_Max_Local_Data_Length) =>MSAC2M_Reject.ind(C_Ref:=Stored_C_Ref, Reason_Code:=REJ_LE)	OPEN
34	OPEN	S-Timer expired =>Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Idle-REQ-PDU, Serv_class:=Low)	IDLE-POLL-RES
35	OPEN	MSAC2M_Abort.req(C_Ref=Stored_C_Ref, Subnet, Instance, Reason_Code) =>Stop S-Timer Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
36	OPEN	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>Stop S-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
37	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = NR =>DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	VALID-SERVICE-POLL-RES
38	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Idle-REQ-PDU =>Stop R-Timer Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Idle-RES-PDU, Serv_class:=Low)	VALID-SERVICE-POLL-RES
39	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = VALID_SERVICE_RES_PDU && Function_Num = Service_Header && Go_Abort = FALSE =>Stop R-Timer Start S-Timer MSAC2M_<VALID_SERVICE>.cnf(+, C_Ref:=Stored_C_Ref, <VALID_SERVICE_RES_PARAM>)	OPEN

#	Current State	Event /Condition =>Actions taken Output	Next State
40	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = VALID_SERVICE_NRS_PDU && Function_Num = (Service_Header+0x80) && Go_Abort = FALSE =>Stop R-Timer Start S-Timer MSAC2M_<VALID_SERVICE>.cnf(-, C_Ref:=Stored_C_Ref, Rem_Add, <VALID_SERVICE_NRS_PARAM>)	OPEN
41	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && NOT(Abort-REQ-PDU) && Go_Abort = TRUE =>ignore L_sdu Stop R-Timer Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU with stored Parameter, Serv_class:=Low)	END-POLL-RES
42	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DS/UE/RS =>Stop R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
43	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && Go_Abort = FALSE && NOT(L_sdu = VALID_SERVICE_RES_PDU && (Function_Num = Service_Header)) && NOT(L_sdu = VALID_SERVICE_NRS_PDU && (Function_Num = (Service_Header+0x80))) && NOT(L_sdu = Abort-REQ-PDU) && NOT(L_sdu = Idle-REQ-PDU) =>Stop R-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_RE) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
44	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = RDL/NA/RR/LR/DH/RDH =>ignore L_sdu if present Stop R-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
45	VALID-SERVICE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Abort-REQ-PDU =>Stop R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=FALSE, Subnet, Instance, Reason_Code) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
46	VALID-SERVICE-POLL-RES	MSAC2M_XXX.req(C_Ref:=Stored_C_Ref, <REQPARAM>) =>MSAC2M_Reject.ind(C_Ref:=Stored_C_Ref, Reason_Code:=REJ_PS)	VALID-SERVICE-POLL-RES
47	VALID-SERVICE-POLL-RES	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	VALID-SERVICE-POLL-RES

#	Current State	Event /Condition =>Actions taken Output	Next State
48	VALID-SERVICE-POLL-RES	MSAC2M_Abort.req(C_Ref=Stored_C_Ref, Subnet, Instance, Reason_Code) =>Go_Abort := TRUE STORE_ABORT_PARAMETER	VALID-SERVICE-POLL-RES
49	VALID-SERVICE-POLL-RES	R-Timer expired =>STORE_ABORT_PARAMETER Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_TO)	END-WAIT-CON
50	END-WAIT-CON	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /(L_status = DL && L_sdu <> Abort-REQ-PDU) (L_status = NR) =>Stop S-Timer Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU with stored Parameter, Serv_class:=Low)	END-POLL-RES
51	END-WAIT-CON	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DS/UE/RS/RDL/NA/RR/LR/RDH/DH =>Stop S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
52	END-WAIT-CON	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Abort-REQ-PDU =>Stop S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=FALSE, Subnet, Instance, Reason_Code) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
53	END-WAIT-CON	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	END-WAIT-CON
54	END-WAIT-CON	MSAC2M_Abort.req(C_Ref=Stored_C_Ref, Subnet, Instance, Reason_Code) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	END-WAIT-CON
55	END-WAIT-CON	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	END-WAIT-CON
56	END-WAIT-CON	S-Timer expired =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_OC)	FINAL-WAIT-CON
57	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = NR =>DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	IDLE-POLL-RES
58	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Idle-RES-PDU && Service_Buffer = empty && Go_Abort = FALSE =>Stop R-Timer Start S-Timer	OPEN
59	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Idle-RES-PDU && Service_Buffer <> empty && Go_Abort = FALSE =>Service_Header:= Service_Buffer[1] Service_Buffer:=empty Stop R-Timer Start R-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Service_Buffer, Serv_class:=Low)	VALID-SERVICE-POLL-RES

#	Current State	Event /Condition =>Actions taken Output	Next State
60	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Idle-RES-PDU && Go_Abort = TRUE =>Stop R-Timer Start S-Timer DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU with stored Parameter, Serv_class:=Low)	END-POLL-RES
61	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu = Abort-REQ-PDU =>Stop R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=FALSE, Subnet, Instance, Reason_Code) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
62	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL && L_sdu <> Idle-RES-PDU && L_sdu <> Abort-REQ-PDU =>Stop R-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_RE) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
63	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DS/UE/RS =>Stop R-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
64	IDLE-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = FDL/NA/RR/LR/DH/RDH =>ignore L_sdu if present Stop R-Timer Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) DMPMM2_DATA_REPLY.req(SSAP:=Client_SAP, DSAP:=Stored_Server_SAP, Rem_Add:=Server_Add, L_sdu:=Abort-REQ-PDU, Serv_class:=Low)	END-POLL-RES
65	IDLE-POLL-RES	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) /Length <= (Stored_Max_Data_Length) && Length <= (Stored_Max_Local_Data_Length) && Service_Buffer = empty =><LOAD_SERVICE_BUFFER>	IDLE-POLL-RES
66	IDLE-POLL-RES	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) /Length > (Stored_Max_Data_Length) Length > (Stored_Max_Local_Data_Length) && Service_Buffer = empty =>MSAC2M_Reject.ind(C_Ref:=Stored_C_Ref, Reason_Code:=REJ_LE)	IDLE-POLL-RES
67	IDLE-POLL-RES	MSAC2M_XXX.req(C_Ref=Stored_C_Ref, <REQPARAM>) //Service_Buffer <> empty =>MSAC2M_Reject.ind(C_Ref:=Stored_C_Ref, Reason_Code:=REJ_PS)	IDLE-POLL-RES
68	IDLE-POLL-RES	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	IDLE-POLL-RES
69	IDLE-POLL-RES	MSAC2M_Abort.req(C_Ref=Stored_C_Ref, Subnet, Instance, Reason_Code) =>Go_Abort:=TRUE STORE_ABORT_PARAMETER	IDLE-POLL-RES

#	Current State	Event /Condition =>Actions taken Output	Next State
70	IDLE-POLL-RES	R-Timer expired =>STORE_ABORT_PARAMETER Start S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_TO)	END-WAIT-CON
71	END-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = NR =>Stop S-Timer MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
72	END-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DL =>Stop S-Timer ignore L_sdu MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
73	END-POLL-RES	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status = DS/UE/RS/RDL/NA/RR/LR/RDH/DH =>ignore L_sdu if present Stop S-Timer MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=FDL, Reason_Code:=L_status) MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
74	END-POLL-RES	MSAC2M_XXX.req(C_Ref:=Stored_C_Ref, <REQPARAM>) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	END-POLL-RES
75	END-POLL-RES	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_SE)	END-POLL-RES
76	END-POLL-RES	MSAC2M_Abort.req(C_Ref:=Stored_C_Ref, Subnet, Instance, Reason_Code) =>ignore	END-POLL-RES
77	END-POLL-RES	S-Timer expired =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_OC)	FINAL-WAIT-CON
78	FINAL-WAIT-CON	DMPMM2_DATA_REPLY.cnf(SSAP=Client_SAP, DSAP=Stored_Server_SAP, Rem_add=Server_Add, L_sdu, Serv_class=Low, L_Status) /L_status <> IV/LS =>ignore L_sdu if present MSAC2M_Closed.ind(C_Ref:=Stored_C_Ref)	CLOSED
79	FINAL-WAIT-CON	MSAC2M_XXX.req(C_Ref:=Stored_C_Ref, <REQPARAM>) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_OC)	FINAL-WAIT-CON
80	FINAL-WAIT-CON	MSAC2M_Initiate.req(C_Ref, Send_Timeout, Features_Supported, Profile_Features_Supported, Profile_Ident_Number, Add_Addr_Param) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_OC)	FINAL-WAIT-CON
81	FINAL-WAIT-CON	MSAC2M_Abort.req(C_Ref:=Stored_C_Ref, Subnet, Instance, Reason_Code) =>MSAC2M_Abort.ind(C_Ref:=Stored_C_Ref, Locally_Generated:=TRUE, Subnet:=NO, Instance:=MSAC2M, Reason_Code:=ABT_OC)	FINAL-WAIT-CON
82	ANY-STATE	MSAC2M_Reset.req =>Stop R-Timer Stop S-Timer MSAC2M_Reset.cnf	POWER-ON

6.6.10 MMAC2

6.6.10.1 Primitive Definitions

6.6.10.1.1 Primitives Exchanged between MMAC2 and FSPMM2

Table 428 - Primitives issued by FSPMM2 to MMAC2

Primitive Name	Source	Associated Parameters	Functions
MInit MM.req	FSPMM2	Max Local Data Length	
Reset.req	FSPMM2	(none)	
Abort.req	FSPMM2	(none)	
Get Master Diag.req	FSPMM2	Rem Add, MDiag Identifier	
Start Seq.req	FSPMM2	Rem Add, Area Code, Timeout	
Download.req	FSPMM2	Rem Add, Area Code, Add Offset, Data	
Upload.req	FSPMM2	Rem Add, Area Code, Add Offset, Data Len	
End Seq.req	FSPMM2	Rem Add	
Act Param.req	FSPMM2	Rem Add, Area Code, Activate	
Act Para Brct.req	FSPMM2	Rem Add, Area Code	

Table 429 - Primitives issued by MMAC2 to FSPMM2

Primitive Name	Source	Associated Parameters	Functions
MInit MM.cnf	MMAC2		
Reset.cnf	MMAC2	(none)	
Get Master Diag.cnf(+)	MMAC2	Rem Add, Diagnosis Data	
Get Master Diag.cnf(-)	MMAC2	Rem Add, Status	
Start Seq.cnf(+)	MMAC2	Rem Add, Max Len Data Unit	
Start Seq.cnf(-)	MMAC2	Rem Add, Status	
Download.cnf(+)	MMAC2	Rem Add	
Download.cnf(-)	MMAC2	Rem Add, Status	
Upload.cnf(+)	MMAC2	Rem Add, Data	
Upload.cnf(-)	MMAC2	Rem Add, Status	
End Seq.cnf(+)	MMAC2	Rem Add	
End Seq.cnf(-)	MMAC2	Rem Add, Status	
Act Param.cnf(+)	MMAC2	Rem Add	
Act Param.cnf(-)	MMAC2	Rem Add, Status	
Act Para Brct.cnf(+)	MMAC2	Rem Add	
Act Para Brct.cnf(-)	MMAC2	Rem Add, Status	
Abort.ind	MMAC2	Rem Add	
Fault.ind	MMAC2	(none)	

6.6.10.1.2 Parameters of MMAC2 Primitives

The parameters used with the primitives exchanged between MMAC2 and FSPMM2 are described in FAL Type 3 Service Definition IEC 61158-5.

Additionally used parameters are:

Table 430 – Parameters Used with Primitives Exchanged with MMAC2

Parameter Name	Description
Rem_Add	DL Address of the remote station

6.6.10.2 State Machine Description

The state machine for Master-Master communication specifies the requester side. Typically, these are programming or diagnostic devices.

After reset, the MMAC2 changes to the "POWER-ON" state and waits for initialisation by the User.

The state machine resumes as long in the "WAIT-REQ" state as it receives a valid request from the User. After this request was sent, the state machine remains in the "WAIT-RES" state, until the service is terminated successfully or incorrectly.

At any time, maximally one service request is in execution.

Local variables of MMAC2

T1

(Unsigned16)

Time which can elapse at most between a request and the receipt of the associated response.

Server_Add

(Unsigned8)

Address of the DP-Master (Class 1) just communicating with.

Service-Header

(Octet-String of length 4)

Intermediate memory for the first four octets of the last request.

Srv_Ist

This list is used to hold all information which are necessary for the service SAP_ACTIVATE.

Upload_Data_Len

(Unsigned8)

This parameter specifies the length of the requested data.

6.6.10.3 MMAC2 State Table

Table 431 - MMAC2 State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	POWER-ON	MMAC2_Minit_MM.req =>MMAC2_Minit_MM.cnf	WAIT-REQ
2	WAIT-REQ	MMAC2_Get_Master_Diag.req(Rem_Add,Identifier) =>Service_Header[1]:=0x41 Service_Header[2]:=Identifier Server_Add:=Rem_Add Start T1 DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=Service_Header, Serv_class:=Low)	WAIT-RES
3	WAIT-REQ	MMAC2_Start_Seq.req(Rem_Add,Area_Code,Timeout) =>Service_Header[1]:=0x42 Service_Header[2]:=Area_Code Service_Header[3-4]:=Timeout Server_Add:=Rem_Add Start T1 DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=Service_Header, Serv_class:=Low)	WAIT-RES
4	WAIT-REQ	MMAC2_Download.req(Rem_Add,Area_Code,Add_Offset,Data) =>Service_Header[1]:=0x42 Service_Header[2]:=Area_Code Service_Header[3-4]:=Add_Offset Server_Add:=Rem_Add Start T1 DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=(Service_Header,Data), Serv_class:=Low)	WAIT-RES
5	WAIT-REQ	MMAC2_Upload.req(Rem_Add,Area_Code,Add_Offset, Data_Len) =>Service_Header[1]:=0x44 Service_Header[2]:=Area_Code Service_Header[3-4]:=Add_Offset Upload_Data_Len := Data_Len Server_Add:=Rem_Add Start T1 DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=(Service_Header,Data_Len), Serv_class:=Low)	WAIT-RES
6	WAIT-REQ	MMAC2_End_Seq.req(Rem_Add) =>Service_Header[1]:=0x45 Server_Add:=Rem_Add Start T1 DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=Service_Header[1], Serv_class:=Low)	WAIT-RES
7	WAIT-REQ	MMAC2_Act_Param.req(Rem_Add, Area_Code, Activate) =>Service_Header[1]:=0x47 Service_Header[2]:=Area_Code Service_Header[3]:=Activate Server_Add:=Rem_Add Start T1 DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=Service_Header[1-3], Serv_class:=Low)	WAIT-RES
8	WAIT-REQ	MMAC2_Act_Para_Brct.req(Rem_Add, Area_Code) =>DMPMM2_DATA.req(SSAP:=54, DSAP:=54, Rem_add:=Rem_Add, L_sdu:=(0x46,Area_Code), Serv_class:=Low)	WAIT-CON
9	WAIT-CON	MMAC2_XXX.req =>MMAC2_Reject.ind(Rem_Add,Reason_Code:=REJ_CO)	WAIT-CON
10	WAIT-CON	DMPMM2_DATA.cnf(SSAP=54, DSAP=54, Rem_Add, Serv_Class, L_status) /L_status=OK/DS =>MMAC2_Act_Para_Brct.cnf(Rem_Add,Status:=L_status)	WAIT-REQ
11	WAIT-RES	MMAC2_XXX.req =>MMAC2_Reject.ind(Rem_Add,Reason_Code:=REJ_PS)	WAIT-RES
12	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=NR =>DMPMM2_DATA_REPLY.req(SSAP:=54, DSAP:=54, Rem_add:=Server_Add, L_sdu.len:=0, Serv_class:=Low)	WAIT-RES
13	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=IV/LS =>MMAC2_Fault.ind	POWER-ON

#	Current State	Event /Condition =>Actions taken&&Output	Next State
14	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DS/NA/UE/RR/RS && Service_Header[1]=0x41 =>Stop T1 MMAC2_Get_Master_Diag.cnf(Rem_Add, Status:=L_status, Diagnostic_Data:=L_sdu)	WAIT-REQ
15	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DS/NA/UE/RR/RS && Service_Header[1]=0x42 =>Stop T1 MMAC2_Start_Seq.cnf(Rem_Add, Status:=L_status)	WAIT-REQ
16	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DS/NA/UE/RR/RS && Service_Header[1]=0x43 =>Stop T1 MMAC2_Download.cnf(Rem_Add, Status:=L_status)	WAIT-REQ
17	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DS/NA/UE/RR/RS && Service_Header[1]=0x44 =>Stop T1 MMAC2_Upload.cnf(Rem_Add, Status:=L_status)	WAIT-REQ
18	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DS/NA/UE/RR/RS && Service_Header[1]=0x45 =>Stop T1 MMAC2_End_Seq.cnf(Rem_Add, Status:=L_status)	WAIT-REQ
19	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DS/NA/UE/RR/RS && Service_Header[1]=0x47 =>Stop T1 MMAC2_Act_Param.cnf(Rem_Add, Status:=L_status)	WAIT-REQ
20	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=RDL/RDH && Service_Header[1]=0x41 =>Stop T1 MMAC2_Get_Master_Diag.cnf(Rem_Add, Status:=RR)	WAIT-REQ
21	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=RDL/RDH && Service_Header[1]=0x42 =>Stop T1 MMAC2_Start_Seq.cnf(Rem_Add, Status:=RR)	WAIT-REQ
22	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=RDL/RDH && Service_Header[1]=0x43 =>Stop T1 MMAC2_Download.cnf(Rem_Add, Status:=RR)	WAIT-REQ
23	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=RDL/RDH && Service_Header[1]=0x44 =>Stop T1 MMAC2_Upload.cnf(Rem_Add, Status:=RR)	WAIT-REQ
24	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=RDL/RDH && Service_Header[1]=0x45 =>Stop T1 MMAC2_End_Seq.cnf(Rem_Add, Status:=RR)	WAIT-REQ
25	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=RDL/RDH && Service_Header[1]=0x47 =>Stop T1 MMAC2_Act_Param.cnf(Rem_Add, Status:=RR)	WAIT-REQ
26	WAIT-RES	T1 expired /Service_Header[1]=0x41 =>MMAC2_Get_Master_Diag.cnf(Rem_Add, Status:=TO)	TIMO
27	WAIT-RES	T1 expired /Service_Header[1]=0x42 =>MMAC2_Start_Seq.cnf(Rem_Add, Status:=TO)	TIMO
28	WAIT-RES	T1 expired /Service_Header[1]=0x43 =>MMAC2_Download.cnf(Rem_Add, Status:=TO)	TIMO
29	WAIT-RES	T1 expired /Service_Header[1]=0x44 =>MMAC2_Upload.cnf(Rem_Add, Status:=TO)	TIMO
30	WAIT-RES	T1 expired /Service_Header[1]=0x45 =>MMAC2_End_Seq.cnf(Rem_Add, Status:=TO)	TIMO
31	WAIT-RES	T1 expired /Service_Header[1]=0x47 =>MMAC2_Act_Param.cnf(Rem_Add, Status:=TO)	TIMO

#	Current State	Event /Condition =>Actions taken&&Output	Next State
32	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]=FE/NE/AD/EA/LE/RE/IP && Service_Header[1]=0x41 =>Stop T1 MMAC2_Get_Master_Diag.cnf(Rem_Add, Status:=L_sdu[1])	WAIT-REQ
33	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]=FE/NE/AD/EA/LE/RE/IP/NI/SE/SC && Service_Header[1]=0x42 =>Stop T1 MMAC2_Start_Seq.cnf(Rem_Add, Status:=L_sdu[1])	WAIT-REQ
34	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]=FE/NE/AD/EA/LE/RE/NC/SC/NI && Service_Header[1]=0x43 =>Stop T1 MMAC2_Download.cnf(Rem_Add, Status:=L_sdu[1])	WAIT-REQ
35	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]=FE/NE/AD/EA/LE/RE/NI/SC && Service_Header[1]=0x44 =>Stop T1 MMAC2_Upload.cnf(Rem_Add, Status:=L_sdu[1])	WAIT-REQ
36	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]=FE/NE/AD/EA/LE/RE/NI/SE && Service_Header[1]=0x45 =>Stop T1 MMAC2_End_Seq.cnf(Rem_Add, Status:=L_sdu[1])	WAIT-REQ
37	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]=FE/NE/AD/EA/LE/RE/IP/SC/NI/DI && Service_Header[1]=0x47 =>Stop T1 MMAC2_Act_Param.cnf(Rem_Add, Status:=L_sdu[1])	WAIT-REQ
38	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DH/LR L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/IP && Service_Header[1]=0x41 && (L_sdu.len<3 L_sdu[1-2]<>Service_Header[1-2]) =>Stop T1 MMAC2_Get_Master_Diag.cnf(Rem_Add, Status:=RE)	WAIT-REQ
39	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DH/LR L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/IP/NI/SE/SC && Service_Header[1]=0x42 && (L_sdu.len<5 L_sdu[1-4]<>Service_Header[1-4]) =>Stop T1 MMAC2_Start_Seq.cnf(Rem_Add, Status:=RE)	WAIT-REQ
40	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DH/LR L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/NC/SC/NI && Service_Header[1]=0x43 && (L_sdu[1-4]<>Service_Header[1-4] L_sdu.len<>4) =>Stop T1 MMAC2_Download.cnf(Rem_Add, Status:=RE)	WAIT-REQ
41	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DH/LR L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/NI/SC && Service_Header[1]=0x44 && (L_sdu.len<5 L_sdu[1-4]<>Service_Header[1-4]) L_sdu.len> Upload_Data_Len +4) =>Stop T1 MMAC2_Upload.cnf(Rem_Add, Status:=RE)	WAIT-REQ
42	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DH/LR L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/NI/SE && Service_Header[1]=0x45 && (L_sdu[1]<>Service_Header[1] L_sdu.len<>1) =>Stop T1 MMAC2_End_Seq.cnf(Rem_Add, Status:=RE)	WAIT-REQ

#	Current State	Event /Condition =>Actions taken&&Output	Next State
43	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DH/LR L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/IP/SC/NI/DI && Service_Header[1]=0x47 && (L_sdu[1-3]<>Service_Header[1-3] L_sdu.len<>3) =>Stop T1 MMAC2_Act_Param.cnf(Rem_Add, Status:=RE)	WAIT-REQ
44	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/IP && Service_Header[1]=0x41 && (L_sdu.len>=3 && L_sdu[1-2]=Service_Header[1-2]) =>Stop T1 MMAC2_Get_Master_Diag.cnf(Rem_Add, Status:=OK, Diagnostic_Data:=L_sdu[3-L_sdu.len])	WAIT-REQ
45	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/IP/NI/SE/SC && Service_Header[1]=0x42 && (L_sdu.len=5 && L_sdu[1-4]=Service_Header[1-4]) =>Stop T1 MMAC2_Start_Seq.cnf(Rem_Add, Status:=OK, Max_Length_Data_Unit:=L_sdu[5])	WAIT-REQ
46	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/NC/SC/NI && Service_Header[1]=0x43 && (L_sdu[1-4]=Service_Header[1-4] && L_sdu.len=4) =>Stop T1 MMAC2_Download.cnf(Rem_Add, Status:=OK)	WAIT-REQ
47	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/NI/SC && Service_Header[1]=0x44 && (L_sdu.len>=5 && L_sdu[1-4]=Service_Header[1-4] && L_sdu.len<= Upload_Data_Len +4) =>Stop T1 MMAC2_Upload.cnf(Rem_Add, Status:=OK,Data:=L_sdu[5-L_sdu.len])	WAIT-REQ
48	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/NI/SE && Service_Header[1]=0x45 && (L_sdu[1]=Service_Header[1] && L_sdu.len=1) =>Stop T1 MMAC2_End_Seq.cnf(Rem_Add, Status:=OK)	WAIT-REQ
49	WAIT-RES	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status) /L_status=DL && L_sdu[1]<>FE/NE/AD/EA/LE/RE/IP/SC/NI/DI && Service_Header[1]=0x47 && (L_sdu=Service_Header && L_sdu.len=3) =>Stop T1 MMAC2_Act_Param.cnf(Rem_Add, Status:=OK)	WAIT-REQ
50	TIMO	DMPMM2_DATA_REPLY.cnf(SSAP=54, DSAP=54, Rem_Add, L_sdu, Serv_Class, L_status)	WAIT-REQ
51	TIMO	MMAC2_XXX.req =>MMAC2_Reject.ind(Rem_Add,Reason_Code:=REJ_CO)	TIMO
52	any state	unexpected FDL reaction =>Stop T1 MMAC22_Fault.ind	POWER-ON
53	any state	MMAC2_Reset.req =>Stop T1	WAIT-REQ

6.7 DLL Mapping Protocol Machines (DMPMS)

6.7.1 DMPMS

6.7.1.1 Primitive Definitions

6.7.1.1.1 Primitives Exchanged between DMPMS and FSPMS

Table 432 - Primitives issued by FSPMS to DMPMS

Primitive Name	Source	Associated Parameters	Functions
Sinit DLL.req	FSPMS	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMS	(none)	

Table 433 - Primitives issued by DMPMS to FSPMS

Primitive Name	Source	Associated Parameters	Functions
Sinit DL.cnf	DMPMS	Loc_Station_Address	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	DMPMS	(none)	
Fault.ind	DMPMS	(none)	

6.7.1.1.2 Primitives Exchanged between DMPMS and MSCY1S

Table 434 - Primitives issued by MSCY1S to DMPMS

Primitive Name	Source	Associated Parameters	Functions
Slave Init req	MSCY1S	(none)	Initialising of the resources used by DP
Slave Deact.req	MSCY1S	(none)	Deactivation of LSAPs for MS0, MS1
Enter.req	MSCY1S	(none)	Preparing the DLL resources for Data Exchange
Leave.req	MSCY1S	(none)	Release the DLL resources for Data Exchange
Set minTsdrr.req	MSCY1S	MinTsdrr	Change of the minTsdrr Parameter in DLL
Slave Diag Upd.req	MSCY1S	Diag Data	Setup of the DL update buffer for Diagnosis
Data Exchange Upd.req	MSCY1S	Diag Flag, Inp Data	Transfer of the Inp Data buffer to the DL update buffer
RD Outp Upd.req	MSCY1S	Outp Data	Setup a copy of the Outp Data buffer
RD Inp Upd.req	MSCY1S	Inp Data	Setup a copy of the Inp Data buffer
Get Cfg Upd.req	MSCY1S	Cfg Data	Setting of configuration data

Table 435 - Primitives issued by DMPMS to MSCY1S

Primitive Name	Source	Associated Parameters	Functions
Slave Init.cnf	DMPMS	(none)	Success of Initialisation
Reset.cnf	DMPMS	(none)	Termination of Reset service
Slave Deact.cnf	DMPMS	(none)	Success of deactivation of LSAPs for MS0, MS1
Set Slave Add.ind	DMPMS	New Slave Add, Ident Number, No Add Chg, Rem Slave Data	Change of the DL address of a DP-Slave. Setting of remanent Slave Data.
Slave Diag.ind	DMPMS	Req Add	A DP-Master fetches diagnosis information.
Set Prm.ind	DMPMS	Req Add, Prm Data	Parameter data are delivered to the DP-Slave. The parameterization is first done in the start-up phase and is also while the DP-Slave is in the Data Exchange mode.
Chk Cfg.ind	DMPMS	Req Add, Cfg Data	Received configuration data to the DP-Slave for checking. They contain the format of input and output areas as well as the information about the data consistency.
Data Exchange.ind	DMPMS	Outp Data	Transmits output data to a DP-Slave and requests input data.
Global Control.ind	DMPMS	Control Command, Group Select	Control Command accepted only from DP Master (Class1) to which this Slave is assigned. Master reports his Operation Mode with this command.

6.7.1.1.3 Primitives Exchanged between DMPMS and MSAC1S, MSRM2S, MSAC2S

Table 436 - Primitives issued by MSAC1S, MSRM2S, MSAC2S to DMPMS

Primitive Name	Source	Associated Parameters	Functions
RSAP_ACTIVATE.req	MSAC1S, MSRM2S, MSAC2S	SSAP, Access, L_sdu_length_list, Indication_Mode	Activates local SAP
SAP_DEACTIVATE.req	MSAC1S, MSRM2S, MSAC2S	SSAP	Deactivates local SAP
REPLY_UPDATE.req	MSAC1S, MSRM2S, MSAC2S	SSAP, L_sdu, Serv_class, Transmit	Write Data in the Update Buffer for Transmission

Table 437 - Primitives issued by DMPMS to MSAC1S, MSRM2S, MSAC2S

Primitive Name	Source	Associated Parameters	Functions
RSAP_ACTIVATE.cnf	DMPMS	SSAP, M_Status	Activation finished
SAP_DEACTIVATE.cnf	DMPMS	SSAP, M_Status	Deactivation finished
REPLY_UPDATE.cnf	DMPMS	SSAP, Serv_class, L_status	Update finished
DATA_REPLY.ind	DMPMS	SSAP, DSAP, Rem_Add, L_sdu, Serv_class, Loc_Add, Update_status	Data/Request received

6.7.1.1.4 Primitives Exchanged between DMPMS and DL

Table 438 - Primitives issued by DL to DMPMS

Primitive Name	Source	Associated Parameters	Functions
DL-DATA.ind	DL	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU	Refer to DL Type 3 Service Definition IEC 61158-3
DL-DATA-REPLY.ind	DL	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU, Update_Status	
DL-REPLY-UPDATE.cnf	DL	Service_class, S_SAP_index, DL_status	
DLM-RESET.cnf	DL	DLM_Status	
DLM-SET-VALUE.cnf	DL	Current_value (1 to n), DLM_status	
DLM-EVENT.ind	DL	Event/Fault	
DLSAP-ACTIVATE.cnf	DL	S_SAP_index, DLM_status	
DLSAP-ACTIVATE-RESPONDER.cnf	DL	S_SAP_index, DLM_status	
DLSAP-DEACTIVATE.cnf	DL	S_SAP_index, DLM_status	

Table 439 - Primitives issued by DMPMS to DL

Primitive Name	Source	Associated Parameters	Functions
DL-REPLY-UPDATE.cnf	DMPMS	Service_class, S_SAP_index, DLSDU, Transmit_strategy	
DLM-RESET.req	DMPMS	(none)	
DLM-SET-VALUE.req	DMPMS	Variable_name(1 to n), Desired_value (1 to n)	
DLM-READ-VALUE.req	DMPMS	Variable_name(1 to n)	
DLSAP-ACTIVATE.req	DMPMS	S_SAP_index, Access, Service_list	
DLSAP-ACTIVATE-RESPONDER.req	DMPMS	S_SAP_index, Access, DLSDU_length_list	
DLSAP-DEACTIVATE.req	DMPMS	S_SAP_index	

6.7.1.1.5 Parameters of DMPMS Primitives

Table 440 – Parameters Used with Primitives Exchanged between DMPMS and FSPMS, MSCY1S, MSAC1S, MSAC2S, MSRM2S

Parameter Name	Description
MasterAdd	This parameter conveys the DL address of the assigned DP-Master.
Input Data Len	This parameter is used to setup the length of Input Data for this Slave.
Output Data Len	This parameter is used to setup the length of Output Data for this Slave.
Min Tsdr	This parameter conveys the value of the DLL variable minTsdr.
Diag Data	This parameter conveys the Diagnosis-RES-PDU
Diag Flag	This parameter indicates that a DP-Slave has new Diag Data
Inp Data	This parameter conveys the DataExchange-RES-PDU
Outp Data	This parameter conveys the DataExchange-REQ-PDU
Cfg Data	This parameter conveys the Chk_Cfg-REQ-PDU
New Slave Add	This parameter conveys the new DL address for this DP-Slave.
Ident Number	This parameter conveys the device type identifier for this DP-Slave.
No Add Chg	This parameter indicates that there is no further address change allowed.
Rem Slave Data	This parameter conveys remanent parameter for this DP-Slave.
Req Add	This parameter conveys the DL address of the initiator of this service.
Prm Data	This parameter conveys the Set_Prm-REQ-PDU.
Control Command	This parameter conveys the sync, freeze actions and the state of the DP-Master.
Group Select	This parameter determines which groups shall be addressed by this command.
SSAP	Local Identifier for Service Access Point
Access	Defines the use of a SAP for one or several Remote Address (Stations)
L_sdu_length_list	Defines the maximum length of incoming or outgoing L_sdu
Indication Mode	Defines the Mode for Indication of poll-sequences without user data
Serv_class	Indicates priority of the service
M_status	Status of the service execution
L_status	Status of the service execution
Transmit	Defines the operation mode of the update buffer (single = buffer is transmitted once)
Rem_Add	DL Address of the remote station
Loc_Add	DL Address of this station
Update_status	Indicates the transmission of an update buffer

6.7.1.2 State Machine Description

The DMPMS (DL Mapping Protocol Machine) makes the connection between the MS0-, MS1-, MS2-related Protocol Machines and Layer 2. All services are mapped by the DMPMS to the DL and its management services of Layer 2. The DMPMS delivers the necessary Layer 2 parameters of the function call (SSAP, DSAP, Serv_class, ...) for Layer 2, receives the confirmations and indications from Layer 2 and passes them to the MS0-, MS1-, MS2-related Protocol Machines.

a) Local variables

Loc_Station_Address

(Unsigned8)

Local variable for the intermediate storage of the own station address which was delivered with the last Set_Value.

Prev_Diag_Flag

(Boolean)

Indicates the state of the Diag_Flag at the last Data_Exchange_Update.req By means of this flag it is detected whether a high prior update is present.

6.7.1.3 DMPMS State Table

Table 441 - DMPMS State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	PON	DMPMS_SInit_DLL.req(Bus Para) =>DLM_SET_VALUE.req(Variable_name1:= "TS ", Variable_name2:= "Baud_rate ", Variable_name3:= "TSL ", Variable_name4:= "min TSDR ", Desired_Value1:= Bus_Para.TS Desired_Value2:= Bus_Para.Baud_rate, Desired_Value3:= Bus_Para.TSL, Desired_Value4:= 11)	DLM-SET-VALUE
2	DLM-SET-VALUE	DLM_SET_VALUE.cnf(DLM_status(1-5) /DLM_status(1-4)=OK =>no action DMPMS_SInit_DLL.cnf	RUN
3	RUN	DMPMS_Enter.req =>DMPMS_ActivateControl := DMPMS_Enter	DLM-CHECK-RSAP-ACT
4	RUN	DMPMS_Slave_Init.req /ActivateRSAPFromCRLList(DMPMS_ActivateControl:= DMPMS_Init)=TRUE =>DMPMS_ActivateControl := DMPMS_Init DLSAP_ACTIVATE_RESPONDER.req(S_SAP_index:=CREP.DSAP.Access:=CREP.Rem Add, DLSDU_length_list:=BuildRSAPServiceListFromCR(), Indication_mode:=CREP.Indication Mode)	DLM-ACT-RSAP
5	DLM-CHECK-RSAP-ACT	/ActivateRSAPFromCRLList(DMPMS_ActivateControl)=TRUE =>DMPMS_ActivateControl := DMPMS_Init DLSAP_ACTIVATE_RESPONDER.req(S_SAP_index:=CREP.DSAP.Access:=CREP.Rem Add, DLSDU_length_list:=BuildRSAPServiceListFromCR(), Indication_mode:=CREP.Indication Mode)	DLM-ACT-RSAP
6	DLM-CHECK-RSAP-ACT	/ActivateRSAPFromCRLList(DMPMS_ActivateControl)=FALSE && DMPMS_ActivateControl:=DMPMS_Init =>DMPMS_Slave_Init.cnf	RUN
7	DLM-CHECK-RSAP-ACT	/ActivateRSAPFromCRLList(DMPMS_ActivateControl)=FALSE && DMPMS_ActivateControl:=DMPMS_Enter =>DLSAP_ACTIVATE.req(S_SAP_index:=58, Access:=CREP.Rem_Add, (Service_activate:=SDN, Role_in_service:=Responder, DLSDU_length_list:=(0,2,0,0)))	DLM-ACT-SAP
8	DLM-ACT-RSAP	DLSAP_ACTIVATE_RESPONDER.cnf(SSAP, DLM_status) /DLM_status=OK	DLM-CHECK-RSAP-ACT
9	DLM-ACT-SAP	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=OK =>Prev_Diag_Flag := False	RUN
10	RUN	DMPMS_Leave.req =>DMPMS_DeactControl:=DMPMS_Leave	CHECK-SAP-DEACT
11	RUN	DMPMS_Slave_Deact.req =>DMPMS_DeactControl:=DMPMS_Deact	CHECK-SAP-DEACT
12	CHECK-SAP-DEACT	/DeactivateSAPFromCRLList(DMPMS_DeactControl)=TRUE =>DLSAP_DEACTIVATE.req(S_SAP_index:=CREP.DSAP)	SAP-DEACT
13	CHECK-SAP-DEACT	/DeactivateSAPFromCRLList(DMPMS_DeactControl)=FALSE =>DMPMS_Slave_Deact.cnf	RUN
14	SAP-DEACT	DLSAP_DEACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=OK	CHECK-SAP-DEACT
15	RUN	DMPMS_Data_Exchange_Upd.req(Diag_Flag, Inp_Data) /Diag_Flag = False && Prev_Diag_Flag = False && Inp_Data.len > 0 =>DL_REPLY_UPDATE.req(S:SAP_index:=NIL, DLSDU:=Inp_Data, Service_Class:=Low, Transmit_startegy:=Multiple)	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
16	RUN	DMPMS_Data_Exchange_Upd.req(Diag_Flag, Inp_Data) /Diag_Flag = False && Prev_Diag_Flag = False && Inp_Data.len = 0 =>no action	RUN
17	RUN	DMPMS_Data_Exchange_Upd.req(Diag_Flag, Inp_Data) /Diag_Flag = False && Prev_Diag_Flag = True && Inp_Data.len > 0 =>Prev_Diag_Flag := False DL_REPLY_UPDATE.req(S_SAP_index:=NIL, DLSDU:=Inp_Data, Service_Class:=Low, Transmit_strategy:=Multiple) DL_REPLY_UPDATE.req(S_SAP_index:=NIL, DLSDU:=NULL-PDU, Service_Class:=High, Transmit_strategy:=Single)	RUN
18	RUN	DMPMS_Data_Exchange_Upd.req(Diag_Flag, Inp_Data) /Diag_Flag = False && Prev_Diag_Flag = True && Inp_Data.len = 0 =>Prev_Diag_Flag := False DL_REPLY_UPDATE.req(S_SAP_index:=NIL, DLSDU:=NULL-PDU, Service_Class:=High, Transmit_strategy:=Single)	RUN
19	RUN	DMPMS_Data_Exchange_Upd.req(Diag_Flag, Inp_Data) /Diag_Flag = True && Inp_Data.len > 0 =>Prev_Diag_Flag := True DL_REPLY_UPDATE.req(S_SAP_index:=NIL, DLSDU:=Inp_Data, Service_Class:=High, Transmit_strategy:=Multiple)	RUN
20	RUN	DMPMS_Data_Exchange_Upd.req(Diag_Flag, Inp_Data) /Diag_Flag = True && Inp_Data.len = 0 =>Prev_Diag_Flag := True DL_REPLY_UPDATE.req(S_SAP_index:=NIL, DLSDU:=ZERO-PDU, Service_Class:=High, Transmit_strategy:=Multiple)	RUN
21	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=NIL =>DMPMS_Data_Exchange.ind(Outp_Data:=DLSDU)	RUN
22	RUN	DMPMS_Slave_Diag_Upd.req(Diag_Data) =>DL_REPLY_UPDATE.req(S_SAP_index:=60, DLSDU:=Diag_Data, Service_Class:=Low, Transmit_strategy:=Multiple)	RUN
23	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=60 =>DMPMS_Slave_Diag.ind(Req_Add:=S_addr)	RUN
24	RUN	DMPMS_Get_Cfg_Upd.req(Cfg_Data) =>DL_REPLY_UPDATE.req(S_SAP_index:=59, DLSDU:=Cfg_Data, Service_class:=Low, Transmit_strategy:=Multiple)	RUN
25	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=59 =>no action	RUN
26	RUN	DL_DATA.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class) /D_SAP_index=58 && DLSDU = Global_Control-REQ-PDU =>DMPMS_Global_Control.ind(Control_Command:=Global_Control-REQ-PDU.Control_Command, Group_Select:=Global_Control-REQ-PDU.Group_Select)	RUN
27	RUN	DL_DATA.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class) /D_SAP_index=58 && DLSDU <> Global_Control-REQ-PDU =>no action	RUN
28	RUN	DMPMS_RD_Inp_Upd.req(Inp_Data) /Inp_Data.len > 0 =>DL_REPLY_UPDATE.req(S_SAP_index:=56, DLSDU:=Inp_Data, Service_Class:=Low, Transmit_strategy:=Multiple)	RUN
29	RUN	DMPMS_RD_Inp_Upd.req(Inp_Data) /Inp_Data.len = 0 =>no action	RUN
30	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=56 =>no action	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
31	RUN	DMPMS_RD_Outp_Upd.req(Outp_Data) /Outp_Data.len > 0 =>DL_REPLY_UPDATE.req(S_SAP_index:=57, DLSDU:=Outp_Data, Service_Class:=Low, Transmit_strategy:=Multiple)	RUN
32	RUN	DMPMS_RD_Outp_Upd.req(Outp_Data) /Outp_Data.len = 0 =>no action	RUN
33	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=57 =>no action	RUN
34	RUN	DMPMS_RSAP_ACTIVATE.req(SSAP, Access, L_sdu_length_list, Indication_Mode) =>DLSAP_ACTIVATE_RESPONDER.req(S_SAP_index:=SSAP, Access:=Access, DLSDU_length_list:=L_sdu_length_list, Indication_mode:=Indication_Mode)	DLM-PASS-ACT
35	DLM-PASS-ACT	DLSAP_ACTIVATE_RESPONDER.cnf(SSAP, DLM_status) =>DMPMS_RSAP_ACTIVATE.cnf(SSAP:=S_SAP_index, M_Status:=DLM_status)	RUN
36	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index<52 =>DMPMS_DATA_REPLY.ind(SSAP:=S_SAP_index, DSAP:=D_SAP_index, Rem_Add:=D_addr, L_sdu:=DLSDU, Serv_class:=Service_class, Loc_Add:=S_addr, Update_status:=Update_status)	RUN
37	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=55 && DLSDU=Set_Slave_Add-REQ-PDU =>DMPMS_Set_Slave_Add.ind(New_Slave_Add:=Set_Slave_Add-REQ-PDU.New_Slave_Add, Ident_Number:=Set_Slave_Add-REQ-PDU.Ident_Number, No_Add_Chg:=Set_Slave_Add-REQ-PDU.No_Add_Chg, Rem_Slave_Data:=Set_Slave_Add-REQ-PDU.Rem_Slave_Data)	RUN
38	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=55 && DLSDU<>Set_Slave_Add-REQ-PDU =>no action	RUN
39	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=62 =>DMPMS_Chk_Cfg.ind(Req_Add:=S_addr, Cfg_Data:=DLSDU)	RUN
40	RUN	DMPMS_Set_minTsdreq(minTsdreq) /MinTsdreq = 0 =>no action	RUN
41	RUN	DMPMS_Set_minTsdreq(minTsdreq) /minTsdreq > 0 =>DLM_SET_VALUE.req(Variable_name1:="minTSDR", Desired_value1:=minTsdreq)	SMINTSDR
42	SMINTSDR	DLM_SET_VALUE.cnf(DLM_status) /DLM_Status=OK =>no action	RUN
43	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) /D_SAP_index=61 =>DMPMS_Set_Prm.ind(Req_Add:=S_addr, Prm_Data:=DLSDU)	RUN
44	RUN	DMPMS_SAP_DEACTIVATE.req(SSAP) =>DLSAP_DEACTIVATE.req(S_SAP_index:=SSAP)	DLM-PASS-DEACT
45	DLM-PASS-DEACT	DLSAP_DEACTIVATE.cnf(S_SAP_index, DLM_status) =>DMPMS_SAP_DEACTIVATE.cnf(SSAP:=S_SAP_index, M_Status:=DLM_status)	RUN
46	RUN	DMPMS_REPLY_UPDATE.req(SSAP, L_sdu, Serv_class, Transmit) /SSAP<52 =>DL_REPLY_UPDATE.req(S_SAP_index:=SSAP, DLSDU:=L_sdu, Service_class:=Serv_class, Transmit_strategy:=Transmit)	RUN
47	RUN	DL_REPLY_UPDATE.cnf(S_SAP_index, Service_class, DL_status) /S_SAP_index<52 =>DMPMS_REPLY_UPDATE.cnf(SSAP:=S_SAP_index, Serv_class:=Service_class, L_status:=DL_status)	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
48	any state	DMPMS_Reset.req =>DLM_RESET.req	WAIT-RESET
49	WAIT-RESET	DLM_RESET.cnf(DLM_status) /DLM_status=OK =>DMPMS_Reset.cnf	PON
50	RUN	DL_REPLY_UPDATE.cnf(S_SAP_index, Service_class, DL_status) /(S_SAP_index=NIL S_SAP_index=62 S_SAP_index=61 S_SAP_index=60 S_SAP_index=59 S_SAP_index=57 S_SAP_index=56) && DL_status=OK =>no action	RUN
51	any state	DLM_RESET.cnf(DLM_status) /DLM_status=NO/IV =>DMPMS_Fault.ind	PON
52	any state	DLM_SET_VALUE.cnf(DLM_status) /DLM_status=NO/IV =>DMPMS_Fault.ind	PON
53	any state	DLSAP_ACTIVATE_RESPONDER.cnf(SSAP, DLM_status) /DLM_status=NO/IV =>DMPMS_Fault.ind	PON
54	any state	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=NO/IV =>DMPMS_Fault.ind	PON
55	any state	DLSAP_DEACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=NO/IV =>DMPMS_Fault.ind	PON
56	any state	DL_REPLY_UPDATE.cnf(S_SAP_index, Service_class, DL_status) /DL_status=LS/LR/IV =>DMPMS_Fault.ind	PON
57	any state	unexpected DL-primitive /D_SAP_index>54 =>DMPMS_Fault.ind	PON
58	any state	unknown DL-primitive /D_SAP_index>54 =>DMPMS_Fault.ind	PON
59	any state	unexpected DLM-primitive =>DMPMS_Fault.ind	PON
60	any state	unknown DLM-primitive =>DMPMS_Fault.ind	PON

6.7.1.4 Functions

Function Name	Description
DeactivateSAPFromCRLList (DMPMS_DeactControl)	This function checks the CRL DP-Slave if there are CRL entries that belongs to MS0-AR and which SSAP has not been deactivated with the DLM-SAP-DEACTIVATE.req service primitive yet. A) If DMPMS_DeactControl=DMPMS_Deact and if there exist one of the following CRL entries that has not been deactivated yet: Entries with DSAP=NIL or DSAP=58 or DSAP=57 or DSAP=56 or DSAP=62 or DSAP=61 or DSAP=60 or DSAP=59 or DSAP=55 than it returns TRUE and sets the local context according to the current CREP. B) If DMPMS_DeactControl=DMPMS_Leave and if there exist one of the following CRL entries that has not been deactivated yet: Entries with DSAP=NIL or DSAP=58 or DSAP=57 or DSAP=56 than it returns TRUE and sets the local context according to one not activated CREP. C) Otherwise it returns FALSE.
ActivateRSAPFromCRLList (DMPMS_ActivateControl)	This function checks the CRL DP-Slave if there are CRL entries that belongs to MS0-AR and which SSAP has not been activated with the DLM-RSAP-ACTIVATE.req service primitive yet. A) If DMPMS_ActivateControl=DMPMS_Init and if there exist one of the following CRL entries that has

	not been activated yet: Entries with DSAP=62 or DSAP=61 or DSAP=60 or DSAP=59 or DSAP=55 than it returns TRUE and sets the local context according to the current CREP. B) If DMPMS_ActivateControl=DMPMS_Enter and if there exist one of the following CRL entries that has not been activated yet: Entries with DSAP=NIL or DSAP=58 or DSAP=57 or DSAP=56 than it returns TRUE and sets the local context according to one not activated the current CREP. C) Otherwise it returns FALSE.
BuildRSAPServiceListFromCR()	This function builds the Service_list parameter for DL-RSAP-ACTIVATE.req primitive from the current CRL context. It returns DLSDU_length_list.Max_DLSDU_length_req_low = CREP.Max_L_sdu Length Req Low DLSDU_length_list.Max_DLSDU_length_req_high = CREP.Max_L_sdu Length Req High DLSDU_length_list.Max_DLSDU_length_ind/con_low = CREP.Max_L_sdu Length Ind Low DLSDU_length_list.Max_DLSDU_length_ind/con_high = CREP.Max_L_sdu Length Ind High

6.7.2 DMPMM1

6.7.2.1 Primitive Definitions

6.7.2.1.1 Primitives Exchanged between FSPMM1 and DMPMM1

Table 442 - Primitives issued by FSPMM1 to DMPMM1

Primitive Name	Source	Associated Parameters	Functions
MInit DLL.req	FSPMM1	Bus Para	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.req	FSPMM1	(none)	
Global Control.req	FSPMM1	Control Command, Group Select	
Set Bus Par.req	FSPMM1	Bus Para	
Delete SC.req	FSPMM1	Address	
Read Value.req	FSPMM1	Variable	

Table 443 - Primitives issued by DMPMM1 to FSPMM1

Primitive Name	Source	Associated Parameters	Functions
MInit DLL.cnf	DMPMM1	(none)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	DMPMM1	(none)	
Global Control.cnf(+)	DMPMM1	(none)	
Global Control.cnf(-)	DMPMM1	Status	
Set Bus Par.cnf	DMPMM1	Status	
Delete SC.cnf	DMPMM1	Status	
Read Value.cnf	DMPMM1	Value, Status	
Fault.ind	DMPMM1	(none)	
Event.ind	DMPMM1	Event, Add Info	

6.7.2.1.2 Primitives Exchanged between MSCY1M and DMPMM1

Table 444 - Primitives issued by MSCY1M to DMPMM1

Primitive Name	Source	Associated Parameters	Functions
Slave Diag.req	MSCY1M	Rem Add	Fetch Diagnosis of a DP-Slave
Set Prm.req	MSCY1M	Rem Add, Prm Data	Parameter data are delivered to the DP-Slave. The parameterization is first done in the start-up phase and also while the DP-Slave is in the Data Exchange mode.
Chk Cfg.req	MSCY1M	Rem Add, Cfg Data	Send configuration data to the DP-Slave for checking. They contain the format of input and output areas as well as the information about the data consistency.
Data Exchange.req	MSCY1M	Rem Add, Outp Data	Transmits output data to a DP-Slave and requests input data.

Table 445 - Primitives issued by DMPMS to MSCY1S

Primitive Name	Source	Associated Parameters	Functions
Slave Diag.cnf(+)	DMPMS	Rem Add, Diag Data	Get Diagnosis from a DP-Slave.
Slave Diag.cnf(-)	DMPMS	Rem Add, Status	This primitive is used to convey a Slave Diag negative confirmation to DMPMM..
Set Prm.cnf(+)	DMPMS	Rem Add	Parametrisation successfully transmitted, the parameters are not checked.
Set Prm.cnf(-)	DMPMS	Rem Add, Status	This primitive is used to convey a Set Prm (Parameter Setting) negative confirmation to DMPMM.
Chk Cfg.cnf(+)	DMPMS	Rem Add	Configuration data successfully transmitted, the parameters are not checked.
Chk Cfg.cnf(-)	DMPMS	Rem Add, Status	This primitive is used to convey a Chk Cfg (configuration Check) negative confirmation to DMPMM.
Data Exchange.cnf(+)	DMPMS	Rem Add, Diag Flag, Inp Data	Get Input Data and Diag lag from a DP-Slave
Data Exchange.cnf(-)	DMPMS	Rem Add, Status	This primitive is used to convey a Data Exchange negative confirmation to DMPMS.

6.7.2.1.3 Primitives Exchanged between DMPMM1 and MSAL1M, MSAC1M

Table 446 - Primitives issued by MSAL1M, MSAC1M to DMPMM1

Primitive Name	Source	Associated Parameters	Functions
DATA_REPLY.req	MSAL1M, MSAC1M	SSAP, DSAP, Rem_Add, L_sdu, Serv_class	Send Service

Table 447 - Primitives issued by DMPMM1 to MSAL1M, MSAC1M

Primitive Name	Source	Associated Parameters	Functions
DATA_REPLY.cnf	DMPMM1	SSAP, DSAP, Rem_Add, L_sdu, Serv_class, L_status	Send Service executed

6.7.2.1.4 Primitives Exchanged between DMPMM1 and MMAC1**Table 448 - Primitives issued by MMAC1 to DMPMM1**

Primitive Name	Source	Associated Parameters	Functions
RSAP_ACTIVATE.req	MMAC1	SSAP, Access, L_sdu_length_list, Indication_Mode	Activates local SAP
REPLY_UPDATE.req	MMAC1	SSAP, L_sdu, Serv_class, Transmit	Write Data in the Update Buffer for Transmission

Table 449 - Primitives issued by DMPMM1 to MMAC1

Primitive Name	Source	Associated Parameters	Functions
RSAP_ACTIVATE.cnf	DMPMM1	SSAP, M_Status	Activation finished
REPLY_UPDATE.cnf	DMPMM1	SSAP, Serv_class, L_status	Update finished
DATA.ind	DMPMM1	SSAP, DSAP, Rem_Add, L_sdu, Serv_class,	Data/Request received
DATA_REPLY.ind	DMPMM1	SSAP, DSAP, Rem_Add, L_sdu, Serv_class, Update_status	Data/Request received

6.7.2.1.5 Primitives Exchanged between DMPMM1 and DL**Table 450 - Primitives issued by DL to DMPMM1**

Primitive Name	Source	Associated Parameters	Functions
DL-DATA.cnf	DL	Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status	Refer to DL Type 3 Service Definition IEC 61158-3
DL-DATA-REPLY.cnf	DL	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status	
DL-DATA.ind	DL	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU	
DL-DATA-REPLY.ind	DL	Service_class, D_addr, D_SAP_index, S_addr, S_SAP_index, DLSDU, Update_Status	
DL-REPLY-UPDATE.cnf	DL	Service_class, S_SAP_index, DL_status	
DLM-RESET.cnf	DL	DLM_Status	
DLM-SET-VALUE.cnf	DL	Current_value (1 to n), DLM_status	
DLM-EVENT.ind	DL	Event/Fault	

Primitive Name	Source	Associated Parameters	Functions
DLSAP-ACTIVATE.cnf	DL	S_SAP_index, DLM_status	
DLSAP-ACTIVATE-RESPONDER.cnf	DL	S_SAP_index, DLM_status	
DLSAP-DEACTIVATE.cnf	DL	S_SAP_index, DLM_status	

Table 451 - Primitives issued by DMPMM1 to DL

Primitive Name	Source	Associated Parameters	Functions
DL-DATA.req	DMPMM1	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU	Refer to DL Type 3 Service Definition IEC 61158-3
DL-DATA-REPLY.req	DMPMM1	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU	
DL-REPLY-UPDATE.cnf	DMPMM1	Service_class, S_SAP_index, DLSDU, Transmit_strategy	
DLM-RESET.req	DMPMM1	(none)	
DLM-SET-VALUE.req	DMPMM1	Variable_name(1 to n) Desired_value (1 to n)	
DLM-READ-VALUE.req	DMPMM1	Variable_name(1 to n)	
DLSAP-ACTIVATE.req	DMPMM1	S_SAP_index, Access, Service_list	
DLSAP-ACTIVATE-RESPONDER.req	DMPMM1	S_SAP_index, Access, DLSDU_length_list	
DLSAP-DEACTIVATE.req	DMPMM1	S_SAP_index	

6.7.2.1.6 Parameters of DMPMM1 Primitives

Table 452 – Parameters Used with Primitives Exchanged with DMPMM1

Parameter Name	Description
Input Data Len	This parameter is used to setup the length of Input Data for this Slave.
Output Data Len	This parameter is used to setup the length of Output Data for this Slave.
Min Tsdr	This parameter conveys the value of the DLL variable minTsdr.
Rem Add	This parameter conveys the DL address of the receiver of this service.
Diag Data	This parameter conveys the Diagnosis-RES-PDU
Diag Flag	This parameter indicates that a DP-Slave has new Diag Data
Prm Data	This parameter conveys the Set_Prm-REQ-PDU.
Cfg Data	This parameter conveys the Chk_Cfg-REQ-PDU
Inp Data	This parameter conveys the DataExchange-RES-PDU
Outp Data	This parameter conveys the DataExchange-REQ-PDU
Status	This parameter contains the cause of a negative confirmation-
SSAP	Local Identifier for Service Access Point
DSAP	Identifier for the remote Service Access Point
Access	Defines the use of a SAP for one or several Remote Address (Stations)
L_sdu_length_list	Defines the maximum length of incoming or outgoing L_sdu
Indication_Mode	Defines the Mode for Indication of poll-sequences without user data
Serv_class	Indicates priority of the service
M_status	Status of the service execution
L_status	Status of the service execution
L_sdu	Defines the DP PDU to be transmitted
Transmit	Defines the operation mode of the update buffer (single = buffer is transmitted once)
Rem_Add	DL Address of the remote station
Update_status	Indicates the transmission of an update buffer

Table 453 – Possible Values of Status

Value	Meaning	Source
DS	Local-DLL/PHY Entity is not in logical token-ring or disconnected from line	Local DLL
NA	Negative ack, no reaction from remote station	Local DLL
NR	No Response data	Remote DLL
OK	Acknowledgement positive	
RE	Format-Error in a Response-frame	Local DMPM
RR	Resources of the remote-DLL Entity not sufficient or not available	Remote DLL
RS	Service or remote-address at remote-LSAP or remote-LSAP not activated; remote-station is no DP-Station remote-station is not yet ready for these functions remote-station is associated with an other Requester optional service not available	Remote DLL
UE	Remote-MMAC/DLL interface error	Remote DLL

6.7.2.2 State Machine Description

The DMPMM1 makes the connection between the MS0,MS1,MM1,MM2 specific parts and Layer 2. Services are mapped by the DMPMM1 to the DL and management of Layer 2. The DMPMM1 delivers the necessary Layer 2 parameters of the function call (SSAP, DSAP, Serv_class, ...) for Layer 2, receives the confirmations and indications from Layer 2 and passes them to the MS0,MS1,MM1/2-Handler.

The State Machine needs some local variables which are described in the following clause.

Act_Max_Diag_Len

(Unsigned8)

Contains the maximal length of the diagnostic information which can be stored locally.

Range: 6 .. 244

Loc_Station_Address

(Unsigned8)

Local variable for the intermediate storage of the own station address which has been transferred with the last Set_Bus_Par.

Range: 0 .. 125

Srv_Ist

This list is used to store all information which are necessary for the service SAP_ACTIVATE.

Act_msac1m_Max_L_sdu_len

(Unsigned8)

This variable is used to store the actual value of the parameter Max_L_sdu_len locally.

Range: 0 .. 240

6.7.2.3 DMPMM1 State Table

Table 454 - DMPMM1 State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	PON	DMPMM1_Minit_FDL.req(Bus_Para) =>DLM_SET_VALUE.req(Variable_name1:= "TS ", Variable_name2:= "Baud_rate ", Variable_name3:= "TSL ", Variable_name4:= "min TSDR ", Variable_name5:= "max TSDR ", Variable_name6:= "TQUI ", Variable_name7:= "TSET ", Variable_name8:= "TTR ", Variable_name9:= "G ", Variable_name10:= "HSA ", Variable_name11:= "max_retry_limit ", Variable_name12:= "in_ring_desired", Desired_Value1:= Bus_Para.TS Desired_Value2:= Bus_Para.Baud_rate, Desired_Value3:= Bus_Para.TSL, Desired_Value4:= Bus_Para.min TSDR, Desired_Value5:= Bus_Para.max TSDR, Desired_Value6:= Bus_Para.TQUI, Desired_Value7:= Bus_Para.TSET, Desired_Value8:= Bus_Para.TTR, Desired_Value9:= Bus_Para.G, Desired_Value10:= Bus_Para.HSA, Desired_Value11:= Bus_Para.max_retry_limit, Desired_Value12:= TRUE)	SET-VALUE
2	PON	DLM_RESET.cnf(DLM_status) /DLM_status=OK =>DMPMM1_Reset.cnf	PON
3	SET-VALUE	DLM_SET_VALUE.cnf(DLM_status(1-11)) /DLM_status(1-11)=OK =>DLSAP_ACTIVATE.req(S_SAP_index:=SSAP, Access:=All, Service_list:=BuildServiceListFromCR())	SAP-ACT
4	SAP-ACT	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=OK && ActivateSAPFromCRList()==TRUE =>DLSAP_ACTIVATE.req(S_SAP_index:=SSAP, Access:=All, Service_list:=BuildServiceListFromCR())	SAP-ACT
5	SAP-ACT	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=OK && ActivateSAPFromCRList()==FALSE =>DLSAP_ACTIVATE_RESPONDER.req(S_SAP_index:=54, Access:=All, DLSDU_Length_list:=BuildLengthListFromCR())	RSAP-ACT
6	RSAP-ACT	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=OK =>DMPMM2_Minit_FDL.cnf	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
7	RUN	DMPMM1_Set_Bus_Par.req(Bus_Para) =>Loc_Station_Address := DL_ADD DLM_SET_VALUE.req(MSAP_0 , Variable_name1:= "TS ", Variable_name2:= "Baud_rate ", Variable_name3:= "TSL ", Variable_name4:= "min TSDR ", Variable_name5:= "max TSDR ", Variable_name6:= "TQUI ", Variable_name7:= "TSET ", Variable_name8:= "TTR ", Variable_name9:= "G ", Variable_name10:= "HSA ", Variable_name11:= "max_retry_limit ", Variable_name12:= "in_ring_desired", Desired_Value1:= Bus_Para.TS Desired_Value2:= Bus_Para.Baud_rate, Desired_Value3:= Bus_Para.TSL, Desired_Value4:= Bus_Para.min TSDR, Desired_Value5:= Bus_Para.max TSDR, Desired_Value6:= Bus_Para.TQUI, Desired_Value7:= Bus_Para.TSET, Desired_Value8:= Bus_Para.TTR, Desired_Value9:= Bus_Para.G, Desired_Value10:= Bus_Para.HSA, Desired_Value11:= Bus_Para.max_retry_limit, Desired_Value12:= TRUE)	SV1
8	SV1	DLM_SET_VALUE.cnf(DLM_status(1-11)) /DLM_status(1-11)=OK =>DMPMM1_Set_Bus_Par.cnf(+)	RUN
9	RUN	DMPMM1_Delete_SC.req(Address) =>DLM_SET_VALUE.req(Variable_name1:="Frame_Sent_Count", Index1:=Address, Variable_name2:="Error_Count", Index2:=Address, Desired_Value1:=0, Desired_Value2:=0)	SV3
10	SV2	DLM_SET_VALUE.cnf(DLM_status(1-2)) /DLM_status(1-2)=OK =>DMPMM1_Delete_SC.cnf(+)	RUN
11	RUN	DLM_EVENT.ind(Event/Fault, Add_info) =>DMPMM1_Event.ind(Event:=Event/Fault, Add_Info:=Add_info)	RUN
12	RUN	DMPMM1_Read_Value.req(Variable) =>DLM_READ_VALUE.req(Variable_name1:=Variable)	RUN
13	RUN	DLM_READ_VALUE.cnf(DLM_status, Value) =>DMPMM1_Read_Value.cnf(Status:=DLM_status,Value)	RUN
14	RUN	DMPMM1_RSAP_ACTIVATE.req(SSAP, Access, L_sdu_length_list, Indication_Mode) =>DLSAP_ACTIVATE_RESPONDER.req(S_SAP_index:=SSAP, Access:=Access, DLSDU_length_list:= L_sdu_length_list, Indication_mode:= Indication_Mode)	RUN
15	RUN	DLM_RSAP_ACTIVATE.cnf(SSAP, DLM_status) /SSAP = 54 && DLM_status=OK =>DMPMM1_RSAP_ACTIVATE.cnf(SSAP:=SSAP,DLM_status)	RUN
16	RUN	DMPMM1_REPLY_UPDATE.req(SSAP, L_sdu, Serv_class, Transmit) =>DL_REPLY_UPDATE.req(S_SAP_index:=SSAP, DLSDU:=L_sdu, Service_class:=Serv_class, Transmit_strategy:=Transmit)	RUN
17	RUN	DL_REPLY_UPDATE.cnf(S_SAP_index, Service_class, DL_status) =>DMPMM1_REPLY_UPDATE.cnf (SSAP:=S_SAP_index, Serv_class:=Service_class, L_status:=DL_status)	RUN
18	RUN	DMPMM1_DATA_REPLY.req (SSAP, DSAP, Rem_Add, L_sdu, Serv_class) =>DL_DATA_REPLY.req (S_SAP_index:=SSAP, D_SAP_index:=DSAP, S_addr:=Rem_Add, DLSDU:=L_sdu, Service_class:=Serv_class)	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
19	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index<>62 && S_SAP_index<>NIL && DL_status<>LS/IV =>DMPMM1_DATA_REPLY.cnf (SSAP:=S_SAP_index, DSAP:=D_SAP_index, Rem_Add:=S_addr, L_sdu:=DLSDU, Serv_class:=Service_class, L_status:=DL_status)	RUN
20	RUN	DL_DATA.ind(S_SAP_index, D_SAP_index, D_addr, S_Addr, DLSDU, Service_class) =>DMPMM1_DATA.ind (SSAP:=S_SAP_index, DSAP:=D_SAP_index, Rem_Add:=S_addr, L_sdu:=DLSDU, Serv_class:=Service_class)	RUN
21	RUN	DL_DATA_REPLY.ind(S_SAP_index, D_SAP_index, S_addr, D_addr, DLSDU, Service_class, Update_status) =>DMPMM1_DATA_REPLY.ind(SSAP:=S_SAP_index, DSAP:=D_SAP_index, Rem_Add:=S_addr, L_sdu:=DLSDU, Serv_class:=Service_class, Update_status:=Update_status)	RUN
22	RUN	DMPMM1_Data_Exchange.req(Rem_Add, Outp_Data) =>DL_DATA_REPLY.req(S_SAP_index:=NIL, D_SAP_index:=NIL, D_addr:=Rem_Add, DLSDU:=Outp_Data, Service_class:=high)	RUN
23	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=NIL && D_SAP_index=NIL && DL_status=NR =>DMPMM1_Data_Exchange.cnf(+)(Rem_Add:=S_addr, Diag_Flag:=False, Inp_Data.len:=0)	RUN
24	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=NIL && D_SAP_index=NIL && DL_status=DL =>DMPMM1_Data_Exchange.cnf(+)(Rem_Add:=S_addr, Diag_Flag:=False, Inp_Data:=DLSDU)	RUN
25	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=NIL && D_SAP_index=NIL && DL_status=DH =>DMPMM1_Data_Exchange.cnf(+)(Rem_Add:=S_addr, Diag_Flag:=True, Inp_Data:=DLSDU)	RUN
26	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=NIL && D_SAP_index=NIL && DL_status=DS/UE/RS/NA =>DMPMM1_Data_Exchange.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
27	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=NIL && D_SAP_index=NIL && DL_status=RR/RDL/RDH =>DMPMM1_Data_Exchange.cnf(-)(Rem_Add:=S_addr, Status:=RR)	RUN
28	RUN	DMPMM1_Slave_Diag.req(Rem_Add) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=60, S_addr:=Rem_Add, DLSDU:=ZERO_PDU, Service_class:=High)	RUN
29	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DL && DLSDU.len >= 6 && DLSDU.len <= Act_Max_Diag_Len && (Diagnosis-RES-PDU.Diag_Master_Add=255 Diagnosis-RES-PDU.Diag_Master_Add=Loc_Station_Address) =>DMPMM1_Slave_Diag.cnf(+)(Rem_Add:=S_addr, Diag_Data:=DLSDU)	RUN
30	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DL && DLSDU.len > Act_Max_Diag_Len && (Diagnosis-RES-PDU.Diag_Master_Add<>255 && Diagnosis-RES-PDU.Diag_Master_Add<>Loc_Station_Address) =>cut Diagnosis-RES-PDU until Act_Max_Diag_Len Diagnosis-RES-PDU.Station_status_3.Ext_Diag_Overflow:=1 Diagnosis-RES-PDU.Station_status_1.Master_Lock := 1 DMPMM1_Slave_Diag.cnf(+)(Rem_Add:=S_addr, Diag_Data:=Diagnosis-RES-PDU)	RUN
31	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DL && DLSDU.len > Act_Max_Diag_Len && (Diagnosis-RES-PDU.Diag_Master_Add=255 Diagnosis-RES-PDU.Diag_Master_Add=Loc_Station_Address) =>cut Diagnosis-RES-PDU until Act_Max_Diag_Len Diagnosis-RES-PDU.Station_status_3.Ext_Diag_Overflow:=1 DMPMM1_Slave_Diag.cnf(+)(Rem_Add:=S_addr, Diag_Data:=Diagnosis-RES-PDU)	RUN
32	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && L_status=DL && DLSDU.len >= 6 && DLSDU.len <= Act_Max_Diag_Len && (Diagnosis-RES-PDU.Diag_Master_Add<>255 && Diagnosis-RES-PDU.Diag_Master_Add<>Loc_Station_Address) =>Diagnosis-RES-PDU.Station_status_1.Master_Lock := 1 DMPMM1_Slave_Diag.cnf(+)(Rem_Add:=S_addr, Diag_Data:=Diagnosis-RES-PDU)	RUN
33	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DL && DLSDU.len < 6 =>DMPMM1_Slave_Diag.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
34	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DH/RDL/RDH =>DMPMM1_Slave_Diag.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
35	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DS/UE/RS/NA/NR/RR =>DMPMM1_Slave_Diag.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
36	RUN	DMPMM1_Global_Control.req(Rem_Add, Control_Command, Group_Select) =>DL_DATA.req(S_SAP_index:=62, D_SAP_index:=58, D_addr:=Rem_Add, DLSDU:=Global_Control-REQ-PDU, Service_class:=High)	RUN
37	RUN	DL_DATA.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=58 && DL_status=OK/DS =>DMPMM1_Global_Control.cnf(Rem_Add:=S_addr, Status:=DL_status)	RUN
38	ANY-STATE	DL_DATA.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /DL_status=LS/LR/IV =>DMPMM1_Fault.ind	PON
39	RUN	DMPMM1_Chk_Cfg.req(Rem_Add, Cfg_Data) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=62, S_addr:=Rem_Add, DLSDU:=Cfg_Data, Service_class:=High)	RUN
40	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=62 && DL_status=NR =>DMPMM1_Chk_Cfg.cnf(+)(Rem_Add:=S_addr)	RUN
41	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=62 && DL_status=DS/NA/RS/RR/UE =>DMPMM1_Chk_Cfg.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
42	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=62 && DL_status=RDL/RDH/DL/DH =>DMPMM1_Chk_Cfg.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
43	RUN	DMPMM1_Set_Prm.req(Rem_Add, Prm_Data) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=61, S_addr:=Rem_Add, DLSDU:=Prm_Data, Service_class:=High)	RUN
44	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=61 && DL_status=NR =>DMPMM1_Set_Prm.cnf(+)(Rem_Add:=S_addr)	RUN
45	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=61 && DL_status=DS/NA/RS/RR/UE =>DMPMM1_Set_Prm.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
46	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /S_SAP_index=62 && D_SAP_index=61 && DL_status=RDL/RDH/DL/DH =>DMPMM1_Set_Prm.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
47	any state	DMPMM1_Reset.req =>DLM_RESET.req(DLMSAP_0)	PON
48	any state	DLM_RESET.cnf(DLM_status) /DLM_status=OK =>DMPMM1_Reset.cnf	PON
49	any state	DLM_RESET.cnf(DLM_status) /DLM_status=NO/IV =>DMPMM1_Fault.ind	PON
50	any state	DLM_SAP_ACTIVATE.cnf(SSAP, DLM_status) /DLM_status=IV/NO =>DMPMM1_Fault.ind	PON
51	any state	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_Status) /DL_status=LS/IV/LR =>DMPMM1_Fault.ind	PON
52	any state	inadmissible or unknown DL primitive =>DMPMM1_Fault.ind	PON
53	any state	DLM_SET_VALUE.cnf(DLM_status(1-11)) /DLM_status(1-11)<>OK =>DMPMM1_Fault.ind	PON
54	any state	DLM_READ_VALUE.cnf(DLM_status(1)) /DLM_status(1)<>OK =>DMPMM1_Fault.ind	PON

6.7.2.4 Functions

Function Name	Description
ActivateSAPFrom CRList()	This function checks the CRL DP-Master (Class 1) if there are CRL entries which SSAP has not been activated with the DLAP-ACTIVATE.req service primitive yet. A) If there exist one of the following CRL entries that has not been activated yet: Entries with SSAP=NIL or SSAP=62 or SSAP=51 or SSAP=54 than it returns TRUE and sets the local context according to the current CREP. B) Otherwise it returns FALSE.
BuildServiceListFrom CR()	This function builds the Service_list parameter for DLSAP-ACTIVATE.req primitive from the current CRL context. A) If SSAP = NIL(MS0) it returns Service_list.Service_activate[1] = SRD Service_list.Role_in_Service[1] = Initiator Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_low=0 Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_high=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_high=244 B) If SSAP = 62(MS0) it returns Service_list.Service_activate[1] = SRD Service_list.Role_in_Service[1] = Initiator Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_low=0 Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_high=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_high=0 Service_list.Service_activate[2] = SDN Service_list.Role_in_Service[2] = Initiator Service_list.DLSDU_length_list[2].Max_DLSDU_length_req_low=0 Service_list.DLSDU_length_list[2].Max_DLSDU_length_req_high=0 Service_list.DLSDU_length_list[2].Max_DLSDU_length_ind/cnf_low=244 Service_list.DLSDU_length_list[2].Max_DLSDU_length_ind/cnf_high=0
BuildLengthFromCR()	This function builds the Service_list parameter for DLSAP-ACTIVATE-RESPONDER.req primitive from the current CRL context. DLSDU_length_list.Max_DLSDU_length_req_low = 244 DLSDU_length_list.Max_DLSDU_length_req_high = 0 DLSDU_length_list.Max_DLSDU_length_ind_low =244 DLSDU_length_list.Max_DLSDU_length_ind_high = 0

6.7.3 DMPMM2

6.7.3.1 Primitive Definitions

6.7.3.1.1 Primitives Exchanged between FSPMM2 and DMPMM2

Table 455 - Primitives issued by FSPMM2 to DMPMM2

Primitive Name	Source	Associated Parameters	Functions
MInit DLL.req	FSPMM2	Bus Para	Refer to FAL Type 3 Service Definition IEC 61158-5 and IEC 61158-3
Reset.req	FSPMM2	(none)	
Read Slave Diag.req	FSPMM2	Rem Add	
Read Output.req	FSPMM2	Rem Add	
Read Input.req	FSPMM2	Rem Add	
Get Cfg.req	FSPMM2	Rem Add	

Primitive Name	Source	Associated Parameters	Functions
Set Slave Add.req	FSPMM2	Rem Add, New Slave Add, Ident Number, No Add Chg, Rem Slave Data	

Table 456 - Primitives issued by DMPMM2 to FSPMM2

Primitive Name	Source	Associated Parameters	Functions
MInit DLL.cnf	DMPMM2	(non)	Refer to FAL Type 3 Service Definition IEC 61158-5
Reset.cnf	DMPMM2	(none)	
Read Slave Diag.cnf(+)	DMPMM2	Rem Add, Diag Data	
Read Slave Diag.cnf(-)	DMPMM2	Rem Add, Status	
Get Cfg.cnf(+)	DMPMM2	Rem Add, Cfg Data	
Get Cfg.cnf(-)	DMPMM2	Rem Add Status	
Read Output.cnf(+)	DMPMM2	Rem Add, Output Data	
Read Output.cnf(-)	DMPMM2	Rem Add, Status	
Read Input.cnf(+)	DMPMM2	Rem Add, Input Data	
Read Input.cnf(-)	DMPMM2	Rem Add, Status	
Set Slave Add.cnf(+)	DMPMM2	Rem Add	
Set Slave Add.cnf(-)	DMPMM2	Rem Add, Status	
Event.ind	DMPMM2	Event, Add Info	
Fault.ind	DMPMM2	Rem Add	

6.7.3.1.2 Primitives Exchanged between DMPMM2 and MSAC2M**Table 457 - Primitives issued by MSAC2M to DMPMM2**

Primitive Name	Source	Associated Parameters	Functions
DATA_REPLY.req	MSAC2M	SSAP, DSAP, Rem_Add, L_sdu, Serv_class	Send Service

Table 458 - Primitives issued by DMPMM2 to MSAC2M

Primitive Name	Source	Associated Parameters	Functions
DATA_REPLY.cnf	DMPMM2	SSAP, DSAP, Rem_Add, L_sdu, Serv_class, L_status	Send Service executed

6.7.3.1.3 Primitives Exchanged between DMPMM2 and MMAC2

Table 459 - Primitives issued by MMAC2 to DMPMM2

Primitive Name	Source	Associated Parameters	Functions
DATA_REPLY.req	MMAC2	SSAP, DSAP, Rem_Add, L_sdu, Serv_class	Send Service
DATA.req	MMAC2	SSAP, DSAP, Rem_Add, L_sdu, Serv_class	Send Service

Table 460 - Primitives issued by DMPMM2 to MMAC2

Primitive Name	Source	Associated Parameters	Functions
DATA_REPLY.cnf	DMPMM2	SSAP, DSAP, Rem_Add, L_sdu, Serv_class, L_status	Send Service executed
DATA.cnf	DMPMM2	SSAP, DSAP, Rem_Add, Serv_class, L_status	Send Service executed

6.7.3.1.4 Primitives Exchanged between DMPMM2 and DL

Table 461 - Primitives issued by DL to DMPMM2

Primitive Name	Source	Associated Parameters	Functions
DL-DATA.cnf	DL	Service_class, D_addr, D_SAP_index, S_SAP_index, DL_status	Refer to DL Type 3 Service Definition IEC 61158-3
DL-DATA-REPLY.cnf	DL	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU, DL_status	
DLM-RESET.cnf	DL	DLM_Status	
DLM-SET-VALUE.cnf	DL	Current_value (1 to n), DLM_status	
DLM-EVENT.ind	DL	Event/Fault	
DLSAP-ACTIVATE.cnf	DL	S_SAP_index, DLM_status	
DLSAP-DEACTIVATE.cnf	DL	S_SAP_index, DLM_status	

Table 462 - Primitives issued by DMPMM2 to DL

Primitive Name	Source	Associated Parameters	Functions
DL-DATA.req	DMPMM2	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU	Refer to DL Type 3 Service Definition IEC 61158-3
DL-DATA-REPLY.req	DMPMM2	Service_class, D_addr, D_SAP_index, S_SAP_index, DLSDU	
DLM-RESET.req	DMPMM2	(none)	
DLM-SET-VALUE.req	DMPMM2	Variable_name(1 to n), Desired_value (1 to n)	
DLM-READ-VALUE.req	DMPMM2	Variable_name(1 to n)	
DLSAP-ACTIVATE.req	DMPMM2	S_SAP_index, Access, Service_list	
DLSAP-DEACTIVATE.req	DMPMM2	S_SAP_index	

6.7.3.1.5 Parameters of FSPMM2 Primitives

The parameters used with the primitives exchanged between DMPMM2 and FSPMM2 are described in IEC 61158-5.

Additionally used parameters are:

Table 463 – Parameters Used with Primitives Exchanged with DMPMM2

Parameter Name	Description
SSAP	Local Identifier for Service Access Point
DSAP	Identifier for the remote Service Access Point
Serv_class	Indicates priority of the service
L_status	Status of the service execution
L_sdu	Defines the DP PDU to be transmitted
Rem_Add	DL Address of the remote station

6.7.3.2 State Machine Description

The DMPMM2 makes the connection between the MS0,MS2,MM1/2 specific parts and Layer 2. Services are mapped by the DMPMM2 to the DL and management of Layer 2. The DMPMM2 delivers the necessary Layer 2 parameters of the function call (SSAP, DSAP, Serv_class, ...) for Layer 2, receives the confirmations and indications from Layer 2 and passes them to the MS0,MS2-Handler.

6.7.3.3 DMPMM2 State Table

Table 464 –DMPMM2 State Table

#	Current State	Event /Condition =>Actions taken&&Output	Next State
1	PON	DMPMM2_Minit_DLL.req(Bus_Para) =>DLM_SET_VALUE.req(Variable_name1:= "TS ", Variable_name2:= "Baud_rate ", Variable_name3:= "TSL ", Variable_name4:= "min TSDR ", Variable_name5:= "max TSDR ", Variable_name6:= "TQUI ", Variable_name7:= "TSET ", Variable_name8:= "TTR ", Variable_name9:= "G ", Variable_name10:= "HSA ", Variable_name11:= "max_retry_limit ", Variable_name12:= "in_ring_desired", Desired_Value1:= Bus_Para.TS Desired_Value2:= Bus_Para.Baud_rate, Desired_Value3:= Bus_Para.TSL, Desired_Value4:= Bus_Para.min TSDR, Desired_Value5:= Bus_Para.max TSDR, Desired_Value6:= Bus_Para.TQUI, Desired_Value7:= Bus_Para.TSET, Desired_Value8:= Bus_Para.TTR, Desired_Value9:= Bus_Para.G, Desired_Value10:= Bus_Para.HSA, Desired_Value11:= Bus_Para.max_retry_limit, Desired_Value12:= TRUE)	SET-VALUE
2	PON	DLM_RESET.cnf(DLM_status) /DLM_status=OK =>DMPMM2_Reset.cnf	PON
3	SET-VALUE	DLM_SET_VALUE.cnf(DLM_status(1-11)) /DLM_status(1-11)=OK =>no action	SAP-CHECK
4	SAP-CHECK	spontaneous /ActivateSAPFromCRList()=TRUE =>DLSAP_ACTIVATE.req(S_SAP_index:=SSAP, Access:=All, Service_list:=BuildServiceListFromCR())	SAP-ACT
5	SAP-CHECK	spontaneous /ActivateSAPFromCRList()=FALSE =>DMPMM2_Minit_DLL.cnf	RUN
6	SAP-ACT	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status=OK =>no action	SAP-CHECK
7	RUN	DLM_EVENT.ind(Event/Fault, Add_info) =>DMPMM2_Event.ind(Event:=Event/Fault, Add_Info:=Add_info)	RUN
8	RUN	DMPMM2_Read_Slave_Diag.req(Rem_Add) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=60, S_addr:=Rem_Add, DLSDU.len:=0, Service_class:=High)	RUN
9	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DL && DLSDU.len >= 6 =>DMPMM2_Read_Slave_Diag.cnf(+)(Rem_Add:=S_addr, Diag_Data:=DLSDU)	RUN
10	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DL && DLSDU < 6 =>DMPMM2_Read_Slave_Diag.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
11	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DH/RDL/RDH =>DMPMM2_Read_Slave_Diag.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
12	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=60 && DL_status=DS/UE/RS/NA/NR/RR =>DMPMM2_Read_Slave_Diag.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
13	RUN	DMPMM2_Get_Cfg.req(Rem_Add) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=59, S_addr:=Rem_Add, DLSDU.len:=0, Service_class:=High)	RUN

#	Current State	Event /Condition =>Actions taken&&Output	Next State
14	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=59 && DL_status=DL =>DMPMM2_Get_Cfg.cnf(+)(Rem_Add:=S_addr, Cfg_Data:=DLSDU)	RUN
15	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=59 && DL_status=DS/UE/RS/NA/NR/RR =>DMPMM2_Get_Cfg.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
16	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=59 && DL_status=DH/RDL/RDH =>DMPMM2_Get_Cfg.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
17	RUN	DMPMM2_Read_Input.req(Rem_Add) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=56, S_addr:=Rem_Add, DLSDU.len:=0, Service_class:=High)	RUN
18	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=56 && DL_status=DL =>DMPMM2_Read_Input.cnf(+)(Rem_Add:=S_addr, Inp_Data:=DLSDU)	RUN
19	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=56 && DL_status=DS/UE/RS/NA/NR/RR =>DMPMM2_Read_Input.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
20	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=56 && DL_status=DH/RDL/RDH =>DMPMM2_Read_Input.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
21	RUN	DMPMM2_Read_Output.req(Rem_Add) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=57, S_addr:=Rem_Add, DLSDU.len:=0, Service_class:=High)	RUN
22	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=57 && DL_status=DL =>DMPMM2_Read_Output.cnf(+)(Rem_Add:=S_addr, Outp_Data:=DLSDU)	RUN
23	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=57 && DL_status=DS/UE/RS/NA/NR/RR =>DMPMM2_Read_Output.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
24	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=57 && DL_status=DH/RDL/RDH =>DMPMM2_Read_Output.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
25	RUN	DMPMM2_Set_Slave_Add.req(Rem_Add, New_Slave_Add, Ident_Number, No_Add_Chg, Rem_Slave_Data) =>DL_DATA_REPLY.req(S_SAP_index:=62, D_SAP_index:=55, S_addr:=Rem_Add, DLSDU:=Set_Slave_Add-REQ-PDU, Service_class:=High)	RUN
26	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=55 && DL_status=NR =>DMPMM2_Set_Slave_Add.cnf(+)(Rem_Add:=S_addr)	RUN
27	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=55 && DL_status=DS/UE/RR/RS/NA =>DMPMM2_Set_Slave_Add.cnf(-)(Rem_Add:=S_addr, Status:=DL_status)	RUN
28	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index=62 && D_SAP_index=55 && DL_status=RDL/RDH/DL/DH =>DMPMM2_Set_Slave_Add.cnf(-)(Rem_Add:=S_addr, Status:=RE)	RUN
29	RUN	DMPMM2_DATA_REPLY.req(SSAP, DSAP, Rem_Add, L_sdu, Serv_class) =>DL_DATA_REPLY.req(S_SAP_index:=SSAP, D_SAP_index:=DSAP, S_addr:=Rem_Add, DLSDU:=L_sdu, Service_class:=Serv_class)	RUN
30	RUN	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /S_SAP_index<>62 && DL_status<>LS/IV/LR =>DMPMM2_DATA_REPLY.cnf(SSAP:=S_SAP_index, DSAP:=D_SAP_index, Rem_Add:=S_addr, L_sdu:=DLSDU, Serv_class:=Service_class, L_status:=DL_status)	RUN
31	RUN	DMPMM2_DATA.req(SSAP, DSAP, Rem_Add, L_sdu, Serv_class) =>DL_DATA.req(S_SAP_index:=SSAP, D_SAP_index:=DSAP, S_addr:=Rem_Add, DLSDU:=L_sdu, Service_class:=Serv_class)	RUN
32	RUN	DL_DATA.cnf(S_SAP_index, D_SAP_index, S_addr, Service_class, DL_status) /DL_status<>LS/IV/LR =>DMPMM2_DATA.cnf(SSAP:=S_SAP_index, DSAP:=D_SAP_index, Rem_Add:=S_addr, Serv_class:=Service_class, L_status:=DL_status)	RUN
33	any state	DLM_RESET.cnf(DLM_status) /DLM_status=NO/IV =>DMPMM2_Fault.ind	PON
34	any state	DLSAP_ACTIVATE.cnf(S_SAP_index, DLM_status) /DLM_status<>OK =>DMPMM2_Fault.ind	PON

#	Current State	Event /Condition =>Actions taken&&Output	Next State
35	any state	DLM_SET_VALUE.cnf(DLM_status(1-11)) /NOT (DLM_status(1-11)=OK) =>DMPMM2_Fault.ind	PON
36	any state	DL_DATA_REPLY.cnf(S_SAP_index, D_SAP_index, S_addr, DLSDU, Service_class, DL_status) /DL_status=LS/IV/LR =>DMPMM2_Fault.ind	RUN
37	any state	inadmissible or unknown DL primitive =>DMPMM2_Fault.ind	RUN
38	any state	DL_DATA.cnf(S_SAP_index, D_SAP_index, S_addr, Service_class, DL_status) /DL_status=LS/IV/LR =>DMPMM2_Fault.ind	PON
39	any state	DMPMM2_Reset.req =>DLM_RESET.req	PON

6.7.3.4 Functions

Table 465 –DMPMM2 Functions

Function Name	Description
ActivateSAPFromCRList()	This function checks the CRL DP-Master (Class 2) if there are CRL entries which SSAP has not been activated with the DLM-SAP-ACTIVATE.req service primitive yet. A) If there exist one of the following CRL entries that has not been activated yet: Entries with SSAP=62 or SSAP=50 or SSAP=54 than it returns TRUE and sets the local context according to the current CREP. B) Otherwise it returns FALSE
BuildServiceListFromCR()	This function builds the Service_list parameter for DLSAP-ACTIVATE.req primitive from the current CRL context. A) If SSAP = 62(MS0) it returns Service_list.Service_activate[1] = SRD Service_list.Role_in_Service[1] = Initiator Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_low=0 Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_high=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_high=0 B) If SSAP = 50(MS2) it returns Service_list.Service_activate[1] = SRD Service_list.Role_in_Service[1] = Initiator Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_high=0 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_high=0 C) If SSAP = 54(MM) it returns Service_list.Service_activate[1] = SRD Service_list.Role_in_Service[1] = Initiator Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_req_high=0 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_low=244 Service_list.DLSDU_length_list[1].Max_DLSDU_length_ind/cnf_high=0 Service_list.Service_activate[2] = SDN

	Service_list.Role_in_Service[2] = Initiator Service_list.DLSDU_length_list[2].Max_DLSDU_length_req_low=2 Service_list.DLSDU_length_list[2].Max_DLSDU_length_req_high=0 Service_list.DLSDU_length_list[2].Max_DLSDU_length_ind/cnf_low=0 Service_list.DLSDU_length_list[2].Max_DLSDU_length_ind/cnf_high=0

6.8 Protocol Options

6.9 Parameters for a DP-Slave

Table 466 - Bus Parameter/Reaction times for a DP-Slave

Baudrate kbit/s	≤187,5	500	1500	3000	6000	12000
Min_Slave_ Interval (ms)	≤ 20	≤ 6	≤ 2	≤ 1,5	≤ 1	≤ 0,6
Send_Timeout (sec*)	4	2	1	1	1	1

*) The parameter Send_Timeout represents a performance feature of a DP-Slave implementation. Each DP-Slave implementation has to ensure that the parameter Send_Timeout reaches the smallest value that is possible.

Another performance feature for a DP-Slave implementation is the time Min_Slave_Interval. Each DP-Slave implementation has to ensure that the Min_Slave_Interval reaches the smallest possible value. This means, that in a DP-System with more than ten stations the Min_Slave_Interval shall not be the dominant factor for the cycle time.

7 Type 4

7.1 FAL Syntax Description

7.1.1 FAL-AR PDU Abstract Syntax

The information stored in an APDU depends on whether the APDU holds a request or a response. The role of the state machine that encodes the APDU (the FSPM) determines how the APDU is encoded.

APDUs always consist of an APDU Header and an APDU Body. In response APDUs the APDU body may be empty.

7.1.1.1 Abstract Syntax of APDU Header

Table 467 defines the contents of the APDU Header.

Table 467 – APDU Header

Field name	Subfield name	Possible values	Constraint (present if)	Comment
ControlStatus	Instruction	Errorcode Write Read And Or Test-And-Set Segmented Read Segmented Write		
ControlStatus	Errorcode	Described in Figure 133 to Figure 135	ControlStatus.Instruction = Errorcode	
ControlStatus	Addressing method	Variable Object Flat	ControlStatus.Instruction <> Errorcode	
ControlStatus	ActualDataError	NoActualError ActualError	ControlStatus.Instruction <> Errorcode	Used by the responding user application to indicate, that an actual error may affect the accessed Variable Object
ControlStatus	HistoricalDataError	NoHistoricalError HistoricalError	ControlStatus.Instruction <> Errorcode	Used by the responding user application to indicate, that an error may have affected the accessed Variable Object
DataFieldFormat	Offset/Attribute	No Offset/Attribute Offset/Attribute		Indicates, whether the APDU Body holds an Offset/Attribute field
DataFieldFormat	Variable Identifier Format	Simple Complex	APDU is a request APDU	Indicates the format of the Variable Identifier in a request APDU
DataFieldFormat	Offset/Attribute size	Integer16 Integer32	APDU is a response APDU AND DataFieldFormat.Offset/Attribute = Offset/Attribute	Indicates the size of the Offset/Attribute field of the APDU Body
DataLength		min. 2		Indicates the total length of the APDU Body. MaxDataSize indicates the max length of the data part of the APDU Body.

7.1.1.2 Abstract Syntax of APDU Body

The APDU Header indicates the interpretation of the contents of the APDU Body.

Table 468 defines the contents of the APDU Body.

Table 468 – APDU Body

Field name	Subfield name	Possible values	Constraint (present if)	Comment
VariableIdentifier	Code.Bitaddressing	No BitAddressing BitAddressing	APDU is a request PDU AND APDU Header indicates Complex VariableIdentifier	If this field indicates BitAddressing, the VariableIdentifier also holds a Bit-no
VariableIdentifier	Code.Bit-no	0-7	APDU is a request PDU AND APDU Header indicates Complex VariableIdentifier AND VariableIdentifier indicates BitAddressing	Bit-no selects a bit within the octet. Bit-no = 0 selects bit 1 etc. The octet is selected by Offset/Attribute.
VariableIdentifier	Code.Offset/Attribute size	Integer16 Integer32	APDU is a request PDU AND DataFieldFormat.Variable Identifier Format = Complex AND DataFieldFormat.Offset/ Attribute = Offset/Attribute	
VariableIdentifier	ID	-32768 - +32767	APDU is a request PDU AND DataFieldFormat.Variable Identifier Format = Simple	
VariableIdentifier	ID	-8388608 - +8388607	APDU is a request PDU AND DataFieldFormat.Variable Identifier Format = Complex	
Offset/Attribute		-32768 - +32767	APDU is a request PDU AND DataFieldFormat.Offset/Attribute = Offset/Attribute AND VariableIdentifier.Code. Offset/Attribute size = Integer16	Negative values select Attribute, positive values select part of constructed variable
Offset/Attribute		-2147483648 - +2147483647	APDU is a request PDU AND DataFieldFormat.Offset/Attribute = Offset/Attribute AND VariableIdentifier.Code. Offset/Attribute size = Integer32	Negative values select Attribute, positive values select part of constructed variable
Offset/Attribute		-32768 - +32767	APDU is a response PDU AND DataFieldFormat.Offset/Attribute = Offset/Attribute AND DataFieldFormat.Offset/ Attribute size = Integer16	Negative values select Attribute, positive values select part of constructed variable
Offset/Attribute		-2147483648 - +2147483647	APDU is a response PDU AND DataFieldFormat.Offset/Attribute = Offset/Attribute AND DataFieldFormat.Offset/ Attribute size = Integer32	Negative values select Attribute, positive values select part of constructed variable
Data		Any		
RequestedLength		0-65535	APDU is a request PDU AND ControlStatus.Instruction indicates Read OR Segmented Read	Indicates the length of Data to Read, as the number of octets
Sequence		0-2	APDU is a request PDU AND ControlStatus.Instruction indicates Segmented Read OR Segmented Write	Indicates whether this request is the first, one in the middle, or the last of a segmented transfer.

7.1.2 Data Types

The notation for data types is the same in Type 4 as for Type 1 for the following types:

- 1) Integer, Integer8, Integer16, Integer32

2) Unsigned, Unsigned8, Unsigned16

3) Floating32, Floating64

7.2 Transfer Syntaxes

7.2.1 APDU Encoding

7.2.1.1 APDU Header Encoding

The abstract syntax of the APDU Header is defined in 7.1.1.1. This subclause describes the encoding of the header. The APDU Header consists of three fields, as shown in Figure 132.

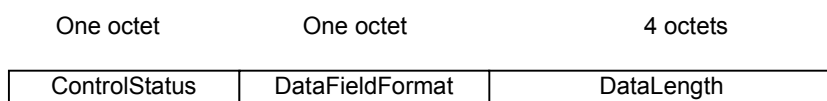


Figure 132 – APDU Header structure

7.2.1.1.1 ControlStatus

ControlStatus is coded into one octet. The interpretation of this octet depends on the instruction subfield. The coding of the instruction subfield is shown in Figure 133.

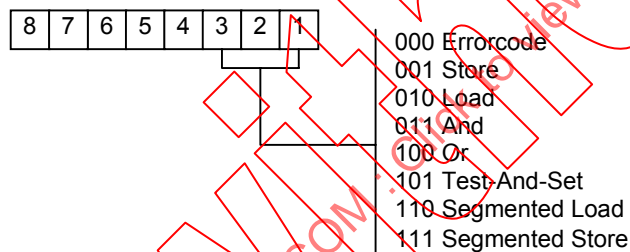


Figure 133 – Instruction subfield of ControlStatus

If the instruction is = 000 (= Errorcode), the remaining five bits of ControlStatus holds the error code. The possible values are shown in Figure 134.

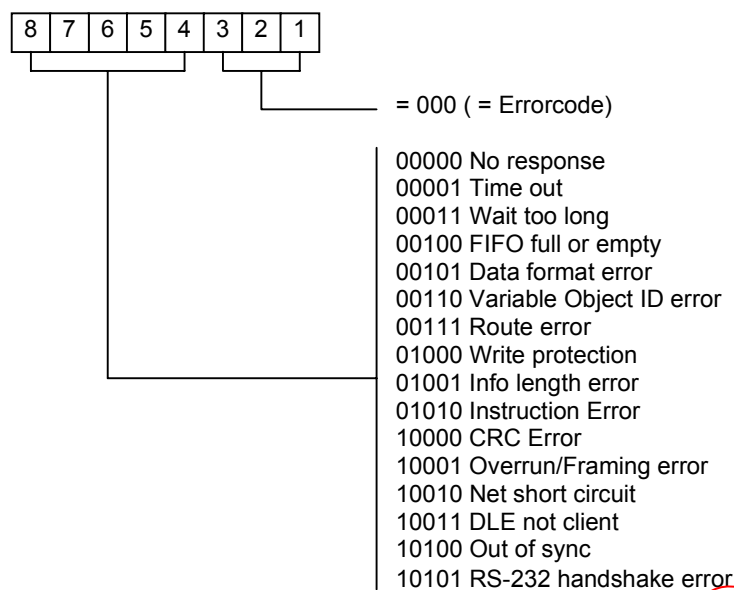


Figure 134 – Errorcode subfield of ControlStatus

If the instruction is $\neq 000$ ($\neq$ Errorcode), the remaining five bits of ControlStatus holds the subfields Addressing method, ActualDataError and HistoricalDataError. The coding of these fields is shown in Figure 135.

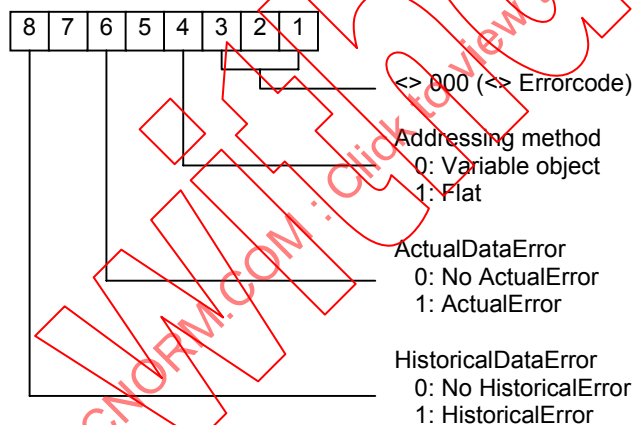


Figure 135 - Remaining subfields of ControlStatus

7.2.1.1.2 DataFieldFormat

DataFieldFormat is coded into one octet. The coding of this octet is shown in Figure 136.

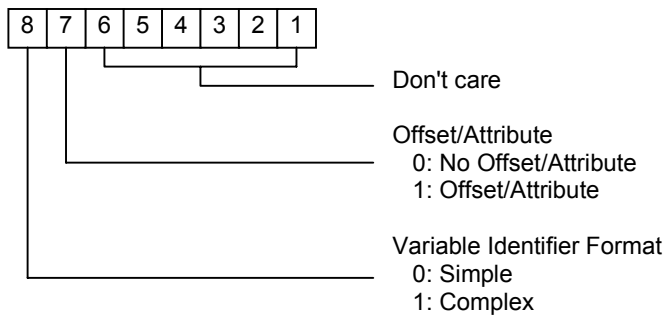


Figure 136 – DataFieldFormat encoding

7.2.1.1.3 DataLength

DataLength is an Integer32, indicating the total length of the APDU Body.

7.2.1.2 APDU Body Encoding

The abstract syntax for the APDU Body is described in 7.1.1.2. This subclause describes the encoding. The interpretation of the APDU Body is indicated by the APDU Header.

A request APDU Body may consist of up to four of five possible fields, as shown in Figure 137.

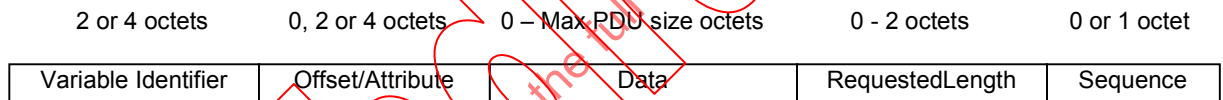


Figure 137 – Structure of request APDU Body

A response APDU Body may consist of up to two fields, as shown in Figure 138.

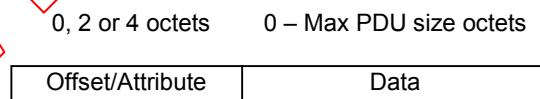


Figure 138 – Structure of response APDU Body

7.2.1.2.1 Variable Identifier

The Variable Identifier can be either Simple or Complex. If it is Simple, it consists of only one subfield, ID, which is of type Integer16. If it is complex, it consists of two subfields, Code (1 octet) and ID (3 octets) as shown in Figure 139.

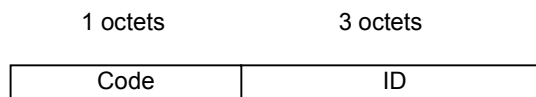
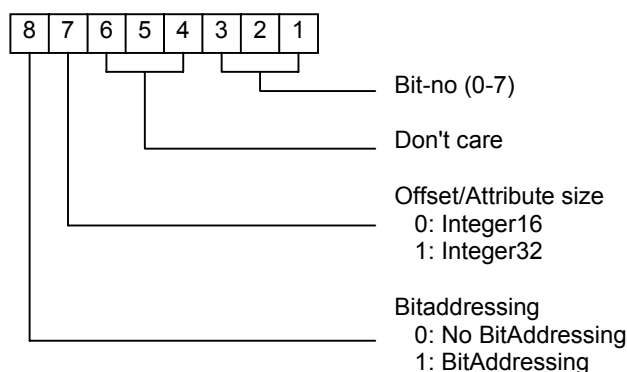


Figure 139 – Variable Identifier

The coding of the Code subfield of the Variable Identifier is shown in Figure 140.

**Figure 140 – Code subfield of Variable Identifier**

7.2.1.2.2 Offset/Attribute

The Offset/Attribute subfield of the APDU Body may or may not be present. If present, Offset/Attribute is an Integer16 or an Integer32. A negative value selects an attribute of the Variable Object. A positive value selects a part of the data value of the Variable Object, by indicating the offset in octets to the starting octet of the data block to be transferred, relative to the first octet of the variable.

7.2.1.2.3 RequestedLength

The RequestedLength subfield may or may not be present.

If the APDU is a request APDU, and the ControlStatus.Instruction subfield of the APDU Header indicates Read or Segmented Read, the RequestedLength subfield is present. It indicates the octet length of the requested data or attribute.

The RequestedLength subfield is an Unsigned8 or an Unsigned16. As there is no Data subfield in the APDU if there is a RequestedLength subfield, the size of RequestedLength is implicitly given by the DataLength parameter. If the value of RequestedLength is less than 256, RequestedLength shall be of type Unsigned8.

7.2.1.2.4 Data

The Data subfield of the APDU Body holds either:

- 1) Data, coded and packed as described in 7.2.2.
- 2) An attribute of a Variable Object, coded and packed as described in 7.2.2.

As there is no RequestedLength subfield in the APDU if there is a Data subfield, the size of the Data subfield is implicitly given by the DataLength parameter.

7.2.1.2.5 Sequence

The Sequence subfield is one octet, with the following legal values:

- 0: Indicating, that this is the first request of a segmented Read or Write.
- 1: Indicating, that this is one of the following requests of a segmented Read or Write.
- 2: Indicating, that this is the last request of a segmented Read or Write.

7.2.2 Variable Object Encoding and Packing

7.2.2.1 Encoding of Simple Variables

7.2.2.1.1 Encoding of a Boolean value

- 1) The encoding of a Boolean value shall be primitive. The ContentsOctets shall consist of a single octet.
- 2) If the Boolean value is FALSE, bit 1 of the ContentsOctets shall be 0 (zero). If the Boolean value is TRUE, bit 1 of the ContentsOctets shall be 1 (one).

7.2.2.1.2 Encoding of an Integer value

As defined in type 1, Transfer syntax 1, Encoding of an Integer Value.

7.2.2.1.3 Encoding of an Unsigned value

As defined in type 1, Transfer syntax 1, Encoding of an Unsigned Value, types Unsigned8 and Unsigned 16.

7.2.2.1.4 Encoding of a Floating Point value

As defined in type 1, Transfer syntax 1, Encoding of a Floating-Point Value.

7.2.2.2 Encoding of Constructed Variables

7.2.2.2.1 Encoding of a String value

- 1) The encoding of a variable length String value shall be primitive.
- 2) The Length field shall indicate as a binary number the number of elements in the String value.
- 3) The Length field shall be in the first octet.

7.2.2.2.2 Encoding of a BitString value

- 1) The encoding of a BitString value shall be primitive.
- 2) There is no Length field in the Type 4 BitString.
- 3) The value of the BitString, commencing with the first bit and proceeding to the trailing bit, shall be placed in bits 1 to 8 of the first octet, followed by bits 1 to 8 of the second octet, followed by bits 1 to 8 of each octet up to and including the last octet of the ContentsOctets.
- 4) Unused bits, if any, shall be placed in bits 2-8 of the last octet.

7.2.2.3 Alignment

This section describes how fields and elements of constructed variables are aligned and transferred.

The general alignment is two.

- Fields and elements of basic type, and with a size of 1 octet, shall be transferred immediately after the previous field or element.
- Fields and elements with a size of more than 1 octet, shall be transferred with an even offset relative to the first octet of the constructed variable.
- Fields and elements of constructed type shall be transferred with an even offset relative to the first octet of the constructed variable.

7.2.2.3.1 Array Elements Transfer Syntax

Table 469 shows an example of transfer syntax for a variable "ArrayVar" of type
ARRAY[1..2] of ARRAY[1..3] of Integer8.

Table 469 – Transfer syntax for Array

1. octet	ArrayVar[1,1]
2. octet	ArrayVar[1,2]
3. octet	ArrayVar[1,3]
4. octet	ArrayVar[2,1]
5. octet	ArrayVar[2,2]
6. octet	ArrayVar[2,3]

7.2.2.3.2 Structure Fields Transfer Syntax

Table 470 shows an example of transfer syntax for a variable "StructVar" of type

STRUCTURE

Field1: Integer8;

Field2: STRUCTURE

Sub1: Integer16;

Sub2: BitString[8];

END;

Field3: Integer8;

Field4: BitString[8];

Field5: Integer8;

END;

Table 470 – Transfer Syntax for Structure

1. octet	StructVar.Field1
2. octet	Dummy
3. octet	StructVar.Field2.Sub1, Most significant octet
4. octet	StructVar. Field2.Sub1, Least significant octet
5. octet	StructVar.Field2.Sub2
6. octet	StructVar.Dummy
7. octet	StructVar.Field3
8. octet	StructVar.Dummy
9. octet	StructVar.Field4
10. octet	StructVar.Field5

7.2.2.4 Variable Object Attributes

All Variable Objects may have the optional attributes defined in Table 471.

Table 471 – Common Variable Object Attributes

Attribute name	Index	Format
Variable Type Identifier	-20	Unsigned8
Octet Length	-24	Integer32
Read enable	-28	Boolean
Write enable	-29	Boolean
Write protected	-30	Boolean

The Key attribute, Variable Identifier, is used to identify the Variable Object, and is not an accessible attribute.

The attributes of a Variable Object may be accessed from the network. If a Variable Object does not support access of an attribute, it shall respond to an access attempt with the Errorcode "Variable Object ID error". Only one attribute can be accessed as a result of one service invocation.

There is one set of attributes for each Variable Object. This means, the subfields or elements of constructed variables do not have their own attributes.

Table 472 shows the variable identifier values for the Variable types defined in Type 4.

Table 472 – Variable Type Identifiers

Variable type	Identifier
Boolean	43
Integer8	49
Integer16	34
Integer32	35
Unsigned8	48
Unsigned16	50
Float32	36
Float64	37
UNICODE Char	51
Complex	61
String	40
BitString	62
FIFO	63

Variable Objects of type FIFO must have the mandatory additional attributes defined in Table 473.

Table 473 – FIFO Variable Object Attributes

Attribute name	Index	Format
Next Element In	-2	Integer16
Next Element Out	-4	Integer16
Free elements	-6	Integer16
Used elements	-8	Integer16
Reread	-9	Boolean
Rewrite	-10	Boolean

7.2.3 Error Codes

This subclause specifies the hexadecimal values for error codes in the error status parameter of the Variable ASE service RESPONSE. The possible values are shown in Table 474.

Table 474 – Error Codes

Error description	Error code (Binary)
Instruction Error	01010000
Info length error	01001000
FIFO full or empty	00100000
Write protection	01000000
Data format error	00101000
Variable Object ID error	00110000
Time Out	00001000
Actual data error	x010xaaa
Historical data error	1xxxxaaa
Route error	00111000
No response	00000000
Wait too long	00011000
Out of sync	10100000
CRC error	10000000
DLE not Client	10011000
Net shortcircuit	10010000
Overrun/Framing error	10001000
RS-232 handshake error	10101000

"x" means the bit is don't care.

"aaa" means at least one of the three bits is true (1).

7.3 FAL Protocol State Machines

Interface to FAL services and protocol machines are specified in the following sections.

The behaviour of the FAL is described by three integrated protocol machines. The three protocol machines are: the FAL Service Protocol Machine (FSPM), the Application Relationship Protocol

Machine (ARPM), and the Data Link Layer Mapping Protocol Machine (DMPM). The relationship among these protocol machines as well as primitives exchanged among them are depicted in Figure 141.

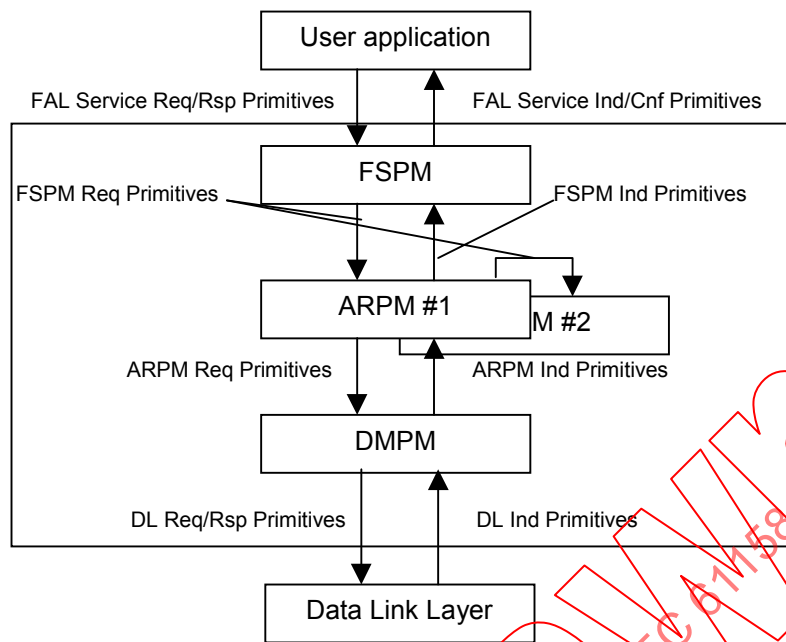


Figure 141 – Summary of FAL Architecture

- The FSPM describes the service interface between the FAL user and a REP. The FSPM is responsible for the following activities:
 - a) To accept service primitives from the FAL service user and convert them into FAL internal primitives.
 - b) To select the AREP identified by the Destination Route parameter supplied by the user application and send FAL internal primitives to the selected AREP.
 - c) To accept FAL internal primitives from the AREP, perform the actions specified in these primitives, and return the result to the AREP from which they are received. This applies for indications holding Request APDUs.
 - d) To accept FAL internal primitives from the AREP, convert them into service, and deliver these service primitives to the FAL user, as a result of the FAL user invocation of the RESPONSE service. This applies for indications holding Response APDUs.
- The ARPM describes exchange of FAL PDUs with remote ARPMs. It does not have any state changes. The ARPM is responsible for the following activities:
 - a) To accept FAL internal primitives from the FSPM and create and send other FAL internal primitives to the DMPM.
 - b) To accept FAL internal primitives from the DMPM and send them to the either the FSPM as indications, or another ARPM as requests.
- The DMPM describes the mapping between the FAL and the DLL. It does not have any state changes. The DMPM is responsible for the following activities:
 - a) To accept FAL internal primitives from the ARPM, prepare DLL service primitives, and send them to the DLL.
 - b) To receive DLL indication primitives from the DLL and send them to the ARPM in a form of FAL internal primitives.

7.4 AP Context State Machine

The AP Context State Machine is not present in Type 4.

7.5 FAL Service Protocol Machine (FSPM)

7.5.1 Primitives Exchanged between FAL User and FSPM

The primitives exchanged between FAL user and FSPM are shown in Table 475.

Table 475 – Primitives Exchanged between FAL-User and FSPM

Primitive name	Source	Associated parameters	Comments
REQUEST.req	FAL user	REP, Variable Object ID, Variable Service, Data length, Offset/Attribute, Bit-no, Data	This primitive is used to convey a FAL user request to a REP
RESPONSE.cnf	FSPM	REP, Variable Service, Data length, Error status, Data	This primitive is used to convey a response from a REP to the FAL user

7.5.2 FSPM States

The following states are defined for an REP: IDLE RESERVED, WAITING FOR RESPONSE, RESPONSE RECEIVED, NOT IN USE.

An REP can be in the role of Proxy object (Client) or Real object (Server). As the behaviour is very different for Proxy object REPs and Real object REPs, they are described in separate subclauses.

7.5.2.1 FSPM Proxy Object States

7.5.2.1.1 Overview

The following states apply to a REP in the Proxy object role:

NOT IN USE

The REP is currently not in use. The only service primitive allowed is ReserveREP.req. All other primitives shall be rejected.

IDLE RESERVED

The REP is reserved by the FAL user, but is currently not active. Allowed service primitives in this state are REQUEST.req, Get REP Attribute.req, Set REP Attribute.req and Free REP.req. All other service primitives shall be rejected.

WAITING FOR RESPONSE

The REP has initiated a transmission, and is monitoring this by a timer. The only service primitive allowed is AR Send.ind from the ARPM, holding the response. All other service primitives shall be rejected.

RESPONSE RECEIVED

The REP has received a response (by an AR Send.ind from the ARPM), or has timed out, and is ready to deliver the response to the FAL user by a RESPONSE.cnf service primitive. Allowed service primitives in this state are REQUEST.req, Get REP Attribute.req, Set REP Attribute.req and Free REP.req from the FAL user. All other service primitives shall be rejected.

Figure 142 depicts the FSP Proxy object state machine.

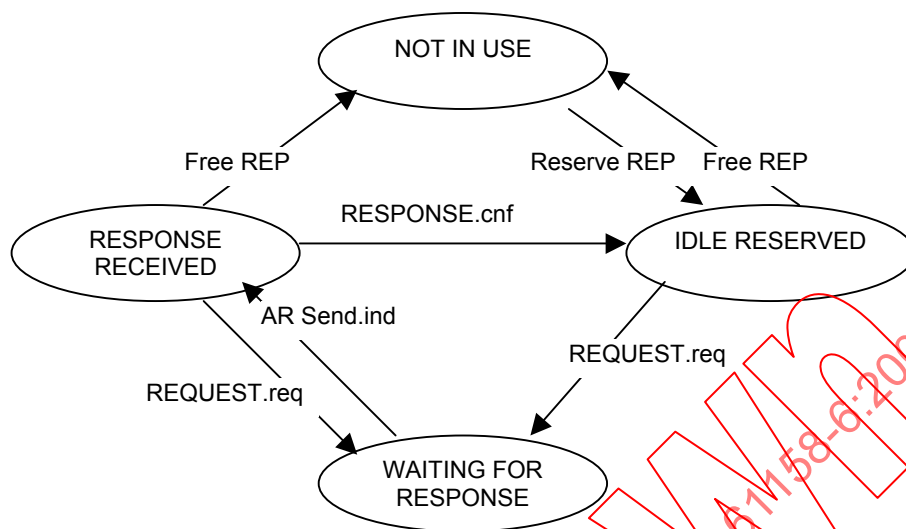


Figure 142 – FSPM Proxy Object State Machine

7.5.2.1.2 Sender State Transitions

7.5.2.1.2.1 REQUEST.req

As a result of this service invocation, the FSPM shall build APDU Header, APDU Body and Route Information, and send them to the ARPM in the form of an AR Send primitive. If anything fails, the request is rejected by returning REQUEST result FAILURE. Constraints are listed in Table 476.

Table 476 – REQUEST.req FSPM Constraints

Condition ID	Condition description
1	REP Attribute State must be IDLE/RESERVED or RESPONSE RECEIVED
2	REP Attribute Role must be Proxy object
3	If REP attribute Confirmation indicates Confirmed, no elements of REP attribute Destination Route may hold the Broadcast address (126)
4	First element of REP attribute Destination Route must hold the address of an AREP in state OPEN
5	If parameter Data length exceeds addressed AREP attribute MaxDataSize, Variable Service may only be Read or Write
6	If parameter Variable Service is Test-And-Set, Data length must be 1
7	If parameter Bit-no indicates bit addressing, Data length must be 1, and parameter Variable Service must be Read, Write or Test-And-Set

If all of the above is fulfilled, the AR Send service primitives are built as described in Table 477.

Table 477 – REQUEST.req FSPM Actions

Action ID	Action description
1	The REP attribute Destination Route is copied to Route info.Destination Route
2	The REP attribute Source Route is copied to Route info.Source Route
3	The REP attribute Endpoint Address is appended to Route info.Source Route
4	If the REP attribute Confirmation indicates Unconfirmed, and no elements of Destination route holds the Broadcast node address (126), the No response node address (0) is appended to Route info.Source Route
5	The REP attribute Priority is copied to Route info.Priority
6	The REP attribute Remote LAN Server ID (if present) is copied to Route info. Remote LAN Server ID
7	Set internal variable Local Bit-no to indicate not bit-addressing, and copy parameter Data length to local variable Data length
8	Handle bit-addressing as described in the following, if parameter Bit-no indicates bit-addressing, and REP attribute Capabilities indicates bit addressing is illegal : a) Save parameter Bit-no in internal variable Local Bit-no, change parameter Bit-no to indicate not bit-addressing, and increment Local Bit-no by one b) If Variable Service parameter is Write and parameter Data.Bit1 is TRUE then b.1) Set APDU Header.ControlStatus.Instruction to Or, and b.2) clear all bits in parameter Data and set bit indicated by Local Bit-no c) If Variable Service parameter is Write and parameter Data.Bit1 is FALSE then c.1) Set APDU Header.ControlStatus.Instruction to And, and c.2) Set all bits in parameter Data and clear bit indicated by Local Bit-no d) If Variable Service parameter is And, Or or Test-And-Set then d.1) Set APDU Header.ControlStatus.Instruction to And, Or, Test-And-Set respectively, and d.2) rotate parameter Data.Bit1 to position indicated by Local Bit-no e) If Variable Service parameter is Read then e.1) Set APDU Header.ControlStatus.Instruction to Read
9	If parameter Data length exceeds addressed AREP attribute MaxDataSize, and parameter Variable Service is Read, set APDU Header.ControlStatus.Instruction to Segmented Read
10	If parameter Data length exceeds addressed AREP attribute MaxDataSize, and parameter Variable Service is Write, set APDU Header.ControlStatus.Instruction to Segmented Write
11	If actions 8, 9 and 10 do not apply, set then set APDU Header ControlStatus.Instruction according to the following: a) Parameter Variable Service Write -> Write b) Parameter Variable Service Read -> Read c) Parameter Variable Service And -> And d) Parameter Variable Service Or -> Or e) Parameter Variable Service Test-And-Set -> Test-And-Set
12	If REP attribute Flat addressing indicates Flat addressing, set APDU Header ControlStatus.Addressing method to Flat. If not, set to Variable Object addressing
13	If a) parameter Bit-no indicates bit-addressing, or b) the value of parameter Variable Object ID is lower than -32768 or higher than +32767, or c) the value of parameter Offset/Attribute is lower than -32768 or higher than +32767, then set APDU Header DataFieldFormat.Variable Identifier Format to Complex, and set APDU Header Data length to 4. If this does not apply, then set APDU Header DataFieldFormat.Variable Identifier Format to Simple, and set APDU Header Data length to 2.
14	If the value of parameter Offset/Attribute is zero, set APDU Header DataFieldFormat.Offset/Attribute to indicate no Offset/Attribute. If not, set to indicate Offset/Attribute
15	If the value of parameter Offset/Attribute is zero, set APDU Header DataFieldFormat.Offset/Attribute to indicate no Offset/Attribute. If not, set to indicate Offset/Attribute
16	If parameter Bit-no indicates bit-addressing, set APDU Body Variable Identifier.Code to indicate Bit-addressing, and insert Bit-no
17	If the value of parameter Offset/Attribute is lower than -32768 or higher than +32767, set APDU Body Variable Identifier.Code to indicate Integer32 Offset/Attribute size
18	If the value of parameter Offset/Attribute is not zero, and within the range of Integer16, copy lower 2 octets to APDU Body Offset/Attribute, and increment APDU Header Data length by 2. If this does not apply, copy parameter Offset/Attribute to APDU Body Offset/Attribute, and increment APDU Header Data length by 4
19	If APDU Header.ControlStatus.Instruction indicates Segmented Read, set RequestedLength in APDU Body Data first octet to "addressed AREP attribute MaxDataSize", if that value is less than or equal to 256, and set APDU Body Data first 2 octets to "addressed AREP attribute MaxDataSize", if that value is higher than 256. Set next octet of APDU Body Data to 0, indicating that this is the first request of a segmented transaction. Increment APDU Header Data length by 2 or 3, depending on size of RequestedLength. Decrement local variable Data length by "addressed AREP attribute MaxDataSize"
20	If APDU Header.ControlStatus.Instruction indicates Segmented Write, copy the first "addressed AREP attribute MaxDataSize – 2" octets from parameter Data to APDU Body Data. Set next octet of APDU Body

	Data to 0, indicating that this is the first request of a segmented transaction. Increment APDU Header Data length by "addressed AREP attribute MaxDataSize – 1". Decrement local variable Data length by "addressed AREP attribute MaxDataSize – 2"
21	If APDU Header.ControlStatus.Instruction indicates Read, set RequestedLength in APDU Body Data first octet to the value of parameter Data length, and increment APDU Header Data length by 1
22	If APDU Header.ControlStatus.Instruction indicates Write, And, Or or Test-And-Set, copy "parameter Data length" octets from parameter Data to APDU Body Data set, and increment APDU Header Data length by the value of "parameter Data length"
23	Send an AR Send request to the addressed AREP
24	If the request fails, return REQUEST.cnf result FAILURE. If the request succeeds, return REQUEST.cnf result OK
25	If the request succeeds, and should be confirmed, set REP Attribute State to WAITING FOR RESPONSE.
26	If the request succeeds, and should not be confirmed, and is a sequenced transaction, build the next APDU in the sequence. This continues, until the complete operation is performed, or an error occurs

7.5.2.1.3 Receiver State Transitions

7.5.2.1.3.1 RESPONSE.cnf

As a result of this service invocation, the FSPM shall check, if a response has been received from the ARPM (FSPM state RESPONSE RECEIVED). If this is the case, deliver the received Variable Service, Data length, Error status and Data to the FAL user. If this is not the case, deliver the Error status "No response". Constraints are listed in Table 478.

Table 478 – RESPONSE.cnf FSPM Constraints

Condition ID	Condition description
1	REP Attribute State must be RESPONSE RECEIVED

If the above is fulfilled, the RESPONSE.cnf primitives are built as described in Table 479.

Table 479 – RESPONSE.cnf FSPM Actions

Action ID	Action description
1	Copy lower 3 bits of local variable ControlStatus to parameter Variable Service
2	Copy local variable Data length to parameter Data length
3	Copy local variable ControlStatus to parameter Error status
4	Copy local variable Data to parameter Data
5	Set REP Attribute State to IDLE RESERVED

7.5.2.1.3.2

7.5.2.1.3.3 AR Send.ind

As a result of this service invocation, the FSPM shall check, if a response is expected from the ARPM (FSPM state WAITING FOR RESPONSE). If this is the case, either parse the received APDU into local variables and change REP state to RESPONSE RECEIVED, or initiate the next AR Send.req if sequenced transaction. If anything fails, do nothing. Constraints are listed in Table 480.

Table 480 – AR Send.ind proxy FSPM Constraints

Condition ID	Condition description
1	REP Attribute State must be WAITING FOR RESPONSE

If the above is fulfilled, the AR Send indication is handled as described in Table 481.

Table 481 – AR Send.ind proxy FSPM Actions

Action ID	Action description
1	If APDU Header.ControlStatus.Instruction indicates Sequenced Read or Sequenced Write, and this is not the final, build next APDU, and Send a new AR Send request to the addressed AREP. If sequenced Read copy received APDU Body Data to local variable Data.
2	If not sequenced transaction, or sequenced transaction finished, copy received APDU Body Data to local variable Data. If Read or Test-And-Set and bit-addressing, copy received APDU Body Data bit "local variable Bit-no" to bit 1 of local Variable Data. Copy received APDU Header ControlStatus to local variable ControlStatus. Copy accumulated received APDU Header Data length to local variable Data length. Set REP Attribute State to IDLE RESERVED

7.5.2.2 FSPM Real Object State Machine Description

7.5.2.2.1 Overview

The following states apply to a REP in the Real object role:

NOT IN USE

The REP is currently not in use. The only service primitive allowed is ReserveREP.req. All other service primitives shall be rejected. Typically, an REP in the Real object role will never be in this state, but will automatically go into the IDLE RESERVED state.

IDLE RESERVED

Typically the initial state of an REP in the Real object role. The REP is reserved by the FAL user, or entered this state automatically, but is not currently active. Allowed service primitives in this state for Real object role REPs are Get REP Attribute.req, Set REP Attribute.req and Free REP.req from the FAL user, and AR Send.ind from the ARPM. All other service primitives shall be rejected.

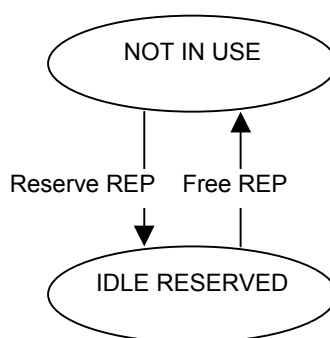
WAITING FOR RESPONSE

An REP in the Real Object role will never go into this state.

RESPONSE RECEIVED

An REP in the Real Object role will never go into this state.

Figure 143 depicts the states of the FSPM Real object state machine.

**Figure 143 – FSPM Real Object State Machine**

7.5.2.2.2 State Transitions

7.5.2.2.2.1 AR Send.ind

As a result of this service invocation, the FSPM shall check, if it is ready to receive an indication from the ARPM (FSPM state IDLE RESERVED). If this is the case, check the received APDU, and if OK interact directly with the addressed Variable Object, which returns the result. Finally, the FSPM shall issue an AR Send.req primitive, to deliver the result. Constraints are listed in Table 482.

Table 482 – AR Send.ind real FSPM Constraints

Condition ID	Condition description
1	REP Attribute State must be IDLE RESERVED
2	APDU Body field Variable Object Identifier must indicate a legal Variable Object

If the above is fulfilled, the AR Send indication is handled as described in Table 483.

Table 483 – AR Send.ind real FSPM Actions

Action ID	Action description
1	If the addressed Variable Object cannot respond within the time specified by AREP Attribute MaxIndicationDelay, issue an AR Acknowledge service primitive. The Destination Route parameter shall be a copy of the Source Route parameter of the AR Send indication. The Source Route parameter shall be a copy of the Destination Route parameter of the AR Send indication. The Priority parameter shall be 0. The Confirmation parameter shall be Unconfirmed. The Remote LAN Server ID parameter shall be a copy of the Remote LAN Server ID parameter of the AR Send indication.
2	Interact directly with the addressed Variable Object, which shall build the appropriate response APDU.
3	Issue an AR Send service primitive. The Destination Route parameter shall be a copy of the Source Route parameter of the AR Send indication. The Source Route parameter shall be a copy of the Destination Route parameter of the AR Send indication. The Priority parameter shall be 0. The Confirmation parameter shall be Unconfirmed. The Remote LAN Server ID parameter shall be a copy of the Remote LAN Server ID parameter of the AR Send indication. APDU Header and APDU Body parameters shall be as build by the Variable Object.

7.6 Application Relationship Protocol Machine (ARPM)

7.6.1 Primitives Exchanged between ARPM and FSPM

The primitives exchanged between ARPM and FSPM, and between one ARPM and another are shown in Table 484 to Table 486.

Table 484 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Comments
AR Send.req	FSPM	Route info, APDU Header, APDU Body	This primitive is used to convey an APDU holding a request or a response from the FSPM to the ARPM
AR Acknowledge.req	FSPM	Route info	This primitive is used by the FSPM to indicate, that the Variable Object is not able to handle the preceding indication immediately

Table 483 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Comments
AR Send.ind	ARPM	Route info, APDU Header, APDU Body	This primitive is used to convey an APDU holding a request or a response from the ARPM to the FSPM

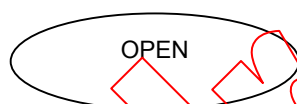
Table 486 – Primitives issued by ARPM to ARPM

Primitive name	Source	Associated parameters	Comments
AR Send.ind	ARPM	Route info, APDU Header, APDU Body	This primitive is used to convey an APDU holding a request or a response from the ARPM to another ARPM

7.6.2 ARPM States

The following states are defined for an AREP: OPEN, as depicted in Figure 144.

7.6.2.1 Overview

**Figure 144 – ARPM State Machine**

OPEN

The AREP is initialised, and ready to accept service primitives.

7.6.2.2 Sender State Transitions

7.6.2.2.1 AR Send.req

As a result of this service invocation, the ARPM shall deliver the received parameters to the addressed DLPM by a new AR Send service invocation. If this is not possible, the request is rejected by returning AR Send result Route error.

Constraints are listed in Table 485.

Table 485 – AR Send.req ARPM Constraints

Condition ID	Condition description
1	AREP Attribute State must be OPEN
2	First address of parameter Route info, Destination route must specify a DLPM in state OPEN

If the above is fulfilled, the AR Send parameters are delivered to the DLPM as described in Table 486.

Table 486 – AR Send.req ARPM Actions

Action ID	Action description
1	Issue an AR Send service primitive to the DLPM, with the same parameters as this indication

7.6.2.2.2 AR Acknowledge.req

As a result of this service invocation, the ARPM shall deliver the received parameters to the addressed DLPM by a new AR Acknowledge service invocation. If this is not possible, the request is rejected.

Constraints are listed in Table 487.

Table 487 – AR Acknowledge.req ARPM Constraints

Condition ID	Condition description
1	AREP Attribute State must be OPEN
2	First address of parameter Route info, Destination route must specify a DLPM in state OPEN

If the above is fulfilled, the AR Acknowledge parameters are delivered to the DLPM as described in Table 488.

Table 488 – AR Acknowledge.req ARPM Actions

Action ID	Action description
1	Issue an AR Acknowledge service primitive to the DLPM, with the same parameters as this indication

7.6.2.3 Receiver State Transitions

7.6.2.3.1 AR Send.ind

As a result of this service invocation, the ARPM shall deliver the received parameters to the addressed destination by a new AR Send service invocation. The addressed destination may be a REP or another AREP. If it is not possible to deliver the received parameters, the request is rejected by returning AR Send result Route error.

Constraints are listed in Table 489.

Table 489 – AR Send.ind ARPM Constraints

Condition ID	Condition description
1	AREP Attribute State must be OPEN
2	First address of parameter Route info, Destination route must either indicate an REP in state IDLE RESERVED or WAITING FOR RESPONSE, or an AREP in state OPEN

If the above is fulfilled, the AR Send parameters are delivered to the FSPM as described in Table 486.

Table 490 – AR Send.req ARPM Actions

Action ID	Action description
1	If the first address of parameter Route info, Destination route indicates an REP, issue an AR Send service primitive to the addressed REP, with the same parameters as this indication
2	If the first address of parameter Route info, Destination route indicates another AREP, issue an AR Send service primitive to the addressed AREP with the same parameters as this indication, and issue an AR Acknowledge primitive to the DLPM from which this indication was received, with the Route info, Destination route parameter equal to the Route info, Source route parameter of this indication, and the Route info, Source route parameter equal to the Route info, Destination route parameter of this indication

7.7 DLL Mapping Protocol Machine (DMPM)

7.7.1 Data Link Layer Service Selection

7.7.1.1 Introduction

This subclause briefly describes the Data Link Layer services utilized by the FAL. These Data Link Layer services are fully defined in the Data Link Layer service specification (IEC 61158-3, Type 4).

7.7.1.1.1 DL-UNITDATA request

This service is used to transmit an APDU from an AREP, and deliver it as a DLSDU to a DLE.

7.7.1.1.2 DL-UNITDATA indication

This service is used to receive a DLSPDU from a DLE and deliver it as an APDU to an AREP.

7.7.1.1.3 DL-UNITDATA response

This service is used to inform the DLE, that a response will not be prepared in time.

7.7.1.1.4 DLM-Set primitive and parameters

This service is used to update the values of attributes of the DLE locally.

7.7.1.1.5 DLM-Get primitive and parameters

This service is used to read the values of attributes of the DLE locally.

7.7.2 Primitives Exchanged between ARPM and DLPM

The primitives exchanged between DLPM and ARPM are shown in Table 491 to Table 492.

Table 491 – Primitives issued by ARPM to DLPM

Primitive name	Source	Associated parameters	Comments
AR Send.req	ARPM	Route info, APDU Header, APDU Body	This primitive is used to convey an APDU holding a request or a response from the ARPM to the DLPM
AR Acknowledge.req	ARPM	Route info	This primitive is used by the ARPM to indicate, that the Variable Object is not able to handle the preceding indication immediately

Table 492 – Primitives issued by DLPM to ARPM

Primitive name	Source	Associated parameters	Comments
AR Send.ind	ARPM	Route info, APDU Header, APDU Body	This primitive is used to convey an APDU holding a request or a response from the DLPM to the ARPM

7.7.3 Primitives Exchanged between DLPM and Data Link Layer

The primitives exchanged between DLPM and ARPM are shown in Table 493 to Table 494.

Table 493 – Primitives issued by DLPM to Data Link Layer

Primitive name	Source	Associated parameters	Comments
DL-UNITDATA request	DLPM	Destination-DL-route, Source-DL-route, Priority, Maximum retry time, Control-status, Data-field-format, DLSDU	This primitive is used to convey a DLSDU holding a request or a response to the DLE
DL-UNITDATA response	DLPM	Destination-DL-route, Source-DL-route	This primitive is used to indicate to the DLE, that the preceding indication cannot be handled immediately

Table 494 – Primitives issued by Data Link Layer to DLPM

Primitive name	Source	Associated parameters	Comments
DL-UNITDATA indication	DLL	Destination-DL-route, Source-DL-route, Confirmation expected, Control-status, Data-field-format, DLSDU	This primitive is used to convey a DLSDU holding a request or a response from the DLE to the DLPM

7.7.4 DLPM States

The following states are defined for a DLPM: OPEN, as depicted in Figure 145.

7.7.4.1 Overview



Figure 145 – DLPM State Machine

OPEN

The DLPM is initialised, and ready to accept service primitives.

7.7.4.2 Sender State Transitions

7.7.4.2.1 AR Send.req

As a result of this service invocation, the DLPM shall deliver the received parameters to the DLE by a DL-UNITDATA request primitive. If this is not possible, the request is rejected.

Constraints are listed in Table 495.

Table 495 – AR Send.req DLPM Constraints

Condition ID	Condition description
1	DLPM state must be OPEN

If the above is fulfilled, the AR Send parameters are converted and delivered to the DLE as described in Table 496.

Table 496 – AR Send.req DLPM Actions

Action ID	Action description
1	First address of parameter Route info, Destination Route, is removed. The result is delivered as the Destination-DL-route parameter of the DL-UNITDATA request primitive
2	Parameter Route info, Source Route is delivered as the Source-DL-route parameter of the DL-UNITDATA request primitive
3	Parameter Route info, Priority is delivered as the Priority parameter of the DL-UNITDATA request primitive
4	AREP attribute MaxRetryTime is delivered as the Maximum retry time parameter of the DL-UNITDATA request primitive
5	Parameter APDU Header, ControlStatus is delivered as the Control-status parameter of the DL-UNITDATA request primitive
6	Parameter APDU Header, DataFieldFormat is delivered in bits 7 and 8, and APDU Header, DataLength is delivered in bits 1 to 6 of the Data-field-format parameter of the DL-UNITDATA request primitive
7	Parameter APDU Body is delivered as the DLSDU parameter of the DL-UNITDATA request primitive

7.7.4.2.2 AR Acknowledge.req

As a result of this service invocation, the DLPM shall deliver the received parameters to the DLE by a DL-UNITDATA response primitive. If this is not possible, no action is taken.

Constraints are listed in Table 497.

Table 497 – AR Acknowledge.req DLPM Constraints

Condition ID	Condition description
1	DLPM state must be OPEN

If the above is fulfilled, the AR Acknowledge parameters are converted and delivered to the DLE as described in Table 498.

Table 498 – AR Acknowledge.req DLPM Actions

Action ID	Action description
1	First address of parameter Route info, Destination Route, is removed. The result is delivered as the Destination-DL-route parameter of the DL-UNITDATA response primitive
2	Parameter Route info, Source Route is delivered as the Source-DL-route parameter of the DL-UNITDATA response primitive

7.7.4.3 Receiver State Transitions**7.7.4.3.1 DL-UNITDATA indication**

As a result of a DL-UNITDATA indication service invocation, the DLPM shall deliver the parameters received from the DLE to the AREP by an AR Send service invocation. If this is not possible, the request is rejected.

Constraints are listed in Table 499.

Table 499 – DL-UNITDATA.ind DLPM Constraints

Condition ID	Condition description
1	AREP Attribute State must be OPEN

If the above is fulfilled, the AR Send parameters are delivered to the ARPM as described in Table 500.

Table 500 – DL-UNITDATA.ind DLPM Actions

Action ID	Action description
1	AREP address is inserted in front of all elements in the parameter Source-DL-route. The result is delivered as the Route info, Source route parameter of the AR Send primitive
2	Parameter Destination-DL-route is delivered as the Route info, Source route parameter of the AR Send primitive
3	Parameter Route info, Priority is set to 0
5	Parameter Confirmation Expected is delivered as the Confirmation parameter of the AR Send primitive
6	Parameter Control-status is delivered as the APDU Header, ControlStatus parameter of the AR Send primitive
7	Bits 1 to 6 of the Data-field-format parameter are delivered as the APDU Header, DataLength parameter of the AR Send primitive
8	Bits 7 and 8 of the Data-field-format parameter are delivered as the APDU Header, DataFieldFormat parameter of the AR Send primitive
9	Parameter DLSDU is delivered as the APDU Body parameter of the AR Send primitive

7.8 Protocol Options

There are no options to select for type 4.

8 Type 5

8.1 FAL Syntax Description

8.1.1 PDU Abstract Syntax

The abstract syntax of Type 5 APDUs is combined with their transfer syntax and is specified below in clause 8.4.

8.1.2 Abstract Syntax of Data Types

The abstract syntax of Type 5 data types is combined with their transfer syntax and is specified below in clause 8.4.

8.2 Transfer Syntax

The Type 5 transfer syntax combines the specification of the abstract syntax and their encodings as a set of fixed format Type 5 APDUs. Each Type 5 APDU contains a header and an APDU body. An optional trailer is also defined. The presence of the trailer is indicated in the Type 5 APDU header.

The format of the Type 5 APDU header and trailer is specified in clauses 8.2.2 and 8.2.2.1.1 below. The APDU header contains the fields common to all messages that are mandatory. The APDU trailer contains the fields common to all APDUs that are optional.

The formats of the Type 5 APDU bodies are specified in clauses 8.2.4. If a service request or response APDU has no parameters, it is stated in the clause that describes the APDU body. If the service has no response or error APDU body associated with it, then the corresponding clause is not present. Common error parameters are used in many service-specific APDUs. Their definitions are specified in clause 8.2.5.1.

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

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

8.2.1 Data Type Encodings

The following data types are used in Type 5 APDUs.

Data Type	Octet Length	Comment
Boolean	1	0 = False, Non-Zero = TRUE
Integer8	1	
Integer16	2	
Unsigned8	1	
Unsigned16	2	
Unsigned32	4	The length of a VisibleString is either fixed by the field definition or it can be dynamically calculated.
VisibleString	-	
OctetString	-	
Time Value	8	The length of an OctetString is either fixed by the field definition or it can be dynamically calculated.

For all data types, the bit and byte order of transmission as defined in Appendix B of RFC 791 – Internet Protocol is used. The bit and byte order specified in RFC 791 Appendix B is the big endian octet ordering (most significant byte first for integers) and bit ordering within bytes is most significant bit first. In the descriptions that follow, bit 1 is the most significant bit in each byte. The unsigned integer value for each bit position is shown in the table below.

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

8.2.2 Type 5 Header

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

All header fields are mandatory.

The format of the header is shown in the table below.

Table 501 – Type 5 APDU Header Format

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Type 5 APDU Version	0	Unsigned8	1	Specifies the Version number of the Type 5 APDU format. The version number of the Type 5 APDUs specified in this document is 1. 0 = Test Version 1 = Version 1
Options	1	Unsigned8	1	Bit 1: 1 = APDU Number present in the Trailer Bit 2: 1 = Invoke Id present in the Trailer (this bit must be set for all client / server Type 5 ARs) Bit 3: 1 = Time Stamp present in the Trailer Bit 4: Extended Control Field present in the Trailer Bit 5: Data Timeliness Flag - user supplied value conveyed transparently. 0 = Not Timely or Not Used 1 = Timely Bits 6-7: Repetition Type 0 = No Repetition 1 = Repeating APDU Body 2 = Repeating Length and APDU Body 3 = Repeating Conveyed Reference, Length, and APDU Body Bit 8: Reserved, set to 0
ASE Id And Confirmed Msg Type	2	Unsigned8	1	Bits 1 – 6: ASE Id - Used to specify the ASE whose service is being conveyed by the APDU. They are interpreted as a 6-bit unsigned integer with the following values: 0 = Unused 1 = AR ASE 2 = SMK ASE 3 = Type 1 ASE and S-VFD Context ASEs 4 = LAN Redundancy ASE 5-63 : Reserved Bits 7, 8: Confirmed Msg Type - Used to differentiate between request, response, and error APDUs. Not used with unconfirmed services. These bits are interpreted as a 2-bit unsigned integer with the following values: 0 = Request APDU 1 = Response APDU 2 = Error APDU 3 = Reserved

Service	3	Unsigned8	1	Specifies the service within the specified protocol. Bit 1: Confirmed Flag (Confirmed=1) and (Unconfirmed=0). When this flag is set for a request APDU, the corresponding response (either a response APDU or an error APDU) is always returned after the request has been processed to indicate the result of the processing. Bits 2-8 Service Id of the service (7-bit unsigned integer). The value used for each Service Id is specified in its APDU Body specification below. It is the number in parenthesis that follows the service name in the section header for each service. All other values are reserved. These values are not necessarily the same as the ASN.1 Tags specified in the abstract syntax for the service.
Conveyed Reference	4	Unsigned32	4	The format and use of the Conveyed Reference is dependent on the ASE Id and the type of S-VFD Context. The next subclause describes this field.
APDU Length	8	Unsigned32	4	Specifies the number of octets contained in the entire APDU, including the header and the trailer.

8.2.2.1 Conveyed Reference Use

This clause specifies the values for the Conveyed Reference field of the header.

Table 502 - Conveyed Reference Use

ASE	S-VFD Context	Bits	Use
S-VFD	N/A	1-32	Service-specific.
SMK	N/A	1-16 17-32	Link Id Set to 0 if the destination SMK is located in the Type 5 device receiving the APDU. If the device is a Linking Device, and the destination SMK is in an Type 1 device attached to the Linking Device, then the Conveyed Reference field contains the link id used to access the Type 1 device. Node.Selector The node value is the high order 8 bits and the selector value is the low order 8 bits, when taken as a 16-bit unsigned integer value. When used to identify a Type 5 SMK (when the Link Id is zero), the node value is set to 0 and the selector value is set to 1. When used to identify an Type 1 SMK (when the Link Id is non-zero), this is the individual or group Type 1 node.selector address. See the description of the Find Tag Query APDU for a detailed description of its use of the Conveyed Reference.
Type 1 ASE/ Type 5 S-VFD	Client/Server	1-16 17-32	Link Id If the server S-VFD is located in the Type 5 device, then this is 0. If the device is a Linking Device, and the destination S-VFD is in an Type 1 device attached to the Linking Device, then this is the Type 1 Link Id of the link to which the Type 1 device is attached. Type 5 Selector Initiate Request: The use of this field is dependent on the Connect Option selected for the request. See the description of the Initiate APDU for the values used. Initiate Response: The Type 5 FAL AE returns the value that is to be used for all subsequent accesses for the S-VFD Context. All Others: The Selector returned in the Initiate Response for access to a specific S-VFD (for both request and response APDUs). NOTE that requesting Type 5 FAL AEs use the Invoke Id to identify the requesting S-VFD Context.
Type 1 ASE/ Type 5 S-VFD	Publisher / Subscriber	1-16 17-32	Link Id If the publisher S-VFD is located in the Type 5 device sending the APDU, then this is 0. If the device is a Linking Device, and the publisher is in a Type 1 device attached to the Linking Device, then this is the Type 1 Link Id of the link to which the publishing device is attached. Dlcep Selector If the publisher is located in the Type 5 device sending the APDU, then this is a flat Dlcep address assigned to the published data. If the device is a Linking Device, and the publisher is in an Type 1 device attached to the Linking Device, then this is Type 1 Dlcep Selector of the publisher buffer.
Type 1 ASE/ Type 5 S-VFD	Report Distribution	1-16 17-32	Link Id If the report source S-VFD is located in the Type 5 device sending the APDU, then this is 0. If the device is a Linking Device, and the report source is in a Type 1 device attached to the Linking Device, then this is the Type 1 Link Id of the link to which the reporting device is attached. S-VFD Context Id or Disap Selector If the report source VFD is located in the Type 5 device sending the APDU, then this is the Report Source S-VFD Context Id. If the device is a Linking Device, and the report source is in an Type 1 device attached to the Linking Device, then this is type 1 Disap Selector of the report source.
NOTE The Link Id, Node, Dlcep Selector, and Disap Selectors are all Type 1 DLL addresses.			

8.2.2.1.1 APDU Sent by a Client S-VFD Endpoint

The following table illustrates the use of the Conveyed Reference header field in APDUs sent by a client S-VFD endpoint.

Message Type	Destination AP	Conveyed Reference	
		Bits	Value
Type 5 Initiate Request APDU Connect Option 1	MIB S-VFD of Type 5 device	Bits 1-16 Bits 17-32	0 (Local Type 5 Device) 0.0
	Non-MIB S-VFD of Type 5 device	Bits 1-16 Bits 17-32	0 (Local Type 5 Device) Context Id Returned by Find Tag
	MIB AP of Linking Device Type 1 Link	Bits 1-16 Bits 17-32	Link Id of Type 1 Link Client Type 1 AP Context Id
	MIB AP of Type 1 device connected to Linking Device	Bits 1-16 Bits 17-32	Link Id of Type 1 Link Client Type 1 AP Context Id
	Non-MIB AP of Type 1 device connected to Linking Device	Bits 1-16 Bits 17-32	Link Id of Type 1 Link Client Type 1 AP Context Id
Type 5 Initiate Request APDU Connect Option 2	MIB S-VFD of Type 5 device	Bits 1-16 Bits 17-32	0 (Local Type 5 Device) 0.0
	MIB AP of Linking Device Type 1 Link	Bits 1-16 Bits 17-32	Link Id of Type 1 Link 0.0
	MIB AP of Type 1 device connected to Linking Device	Bits 1-16 Bits 17-32	Link Id of Type 1 Link Node.0
All Other Request Messages	MIB S-VFD of Type 5 device	Bits 1-32	Conveyed Reference returned in Initiate Response message
	Non- MIB S-VFD	Bits 1-32	Conveyed Reference returned in Initiate Response message
	MIB AP of Linking Device Type 1 Link	Bits 1-32	Conveyed Reference returned in Initiate Response message
	MIB AP of Type 1 device connected to Linking Device	Bits 1-32	Conveyed Reference returned in Initiate Response message
	Non-MIB AP of Type 1 device connected to Linking Device	Bits 1-32	Conveyed Reference returned in Initiate Response message

8.2.2.1.2 APDUs Sent by a Server AR S-VFD Endpoint

The following table illustrates the use of the Conveyed Reference header field in APDUs sent by a server S-VFD endpoint.

Message Type	Destination AP	Bits	Conveyed Reference Value
Type 5 Initiate Response APDU Connect Option 1	MIB S-VFD of Type 5 device	Bits 1-16 Bits 17-32	0 (Local TYPE 5 Device) S-VFD Context Id
	Non- MIB S-VFD of Type 5 device	Bits 1-16 Bits 17-32	Link Id of Type 1 Link S-VFD Context Id
	MIB AP of Linking Device Type 1 Link	Bits 1-16 Bits 17-32	Link Id of Type 1 Link S-VFD Context Id
	MIB AP of Type 1 device connected to Linking Device	Bits 1-16 Bits 17-32	0 (Local Type 5 Device) S-VFD Context Id
	Non-MIB AP of Type 1 device connected to Linking Device	Bits 1-16 Bits 17-32	Link Id of Type 1 Link S-VFD Context Id
Type 5 Initiate Response APDU Connect Option 2	MIB S-VFD of Type 5 device	Bits 1-16 Bits 17-32	0 (Local Type 5 Device) S-VFD Context Id
	MIB AP of Linking Device Type 1 Link	Bits 1-16 Bits 17-32	Link Id of Type 1 Link S-VFD Context Id
	MIB AP of Type 1 device connected to Linking Device	Bits 1-16 Bits 17-32	Link Id of Type 1 Link S-VFD Context Id
All Other Response Messages	MIB S-VFD of Type 5 device	Bits 1-32	Same as Request Message Same as Request Message
	Non- MIB S-VFD of Type 5 device	Bits 1-32	Same as Request Message Same as Request Message
	MIB AP of Linking Device Type 1 Link	Bits 1-32	Same as Request Message Same as Request Message
	MIB AP of Type 1 device connected to Linking Device	Bits 1-32	Same as Request Message Same as Request Message
	Non-MIB AP of Type 1 device connected to Linking Device	Bits 1-32	Same as Request Message Same as Request Message

8.2.2.1.3 APDUs Sent by a Publisher S-VFD Endpoint

The following table illustrates the use of the Conveyed Reference in messages sent by a publisher S-VFD endpoint.

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

8.2.2.1.4 Messages Sent by a Report Source S-VFD Endpoint

The following table illustrates the use of the Conveyed Reference in APDUs sent by a report source S-VFD endpoint.

Message Type	Report Source AP	Conveyed Reference	
		Bits	Value
Type 1 Request Message	AP in the Type 5 device	Bits 1-32	Flat Type 1 address assigned to the report source AP.
	AP of Type 1 device connected through a Type 1 link of a Linking Device	Bits 1-16	Link Id of Type 1 Link
		Bits 17-32	Node.Selector Disap of the source of the report

8.2.3 Type 5 Trailer

Type 5 ARs define which trailer fields are defined for the APDUs, except for SMK ASE APDUs, that they convey. The use of trailer fields by SMK ASE APDUs is service specific. See the SMK ASE APDU definition for their use of trailer fields.

The presence of trailer fields is negotiated for Client / Server sessions and configured for Publisher / Subscriber and Report Distribution Type 5 AR. Their presence is indicated in the Message Header Options field of the each message. The fields that are present are positioned after the user data in the order shown in the table below.

Table 503 – Type 5 APDU Trailer Fields

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

8.2.4 APDU Body

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

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

8.2.4.1 Type 5 AR ASE Services

8.2.4.1.1 Establish Service (Confirmed Service Id = 1)

8.2.4.1.1.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Originator Or Session Identifier	0	Unsigned32	4	Originator Attribute value in request, Application Relationship Identifier in response.
Max Buffer Size	4	Unsigned32	4	Attribute value
Reserved	8	Unsigned8	1	Unused, set to 0.
Configuration Use	9	Unsigned8	1	0 = Configuration Not Permitted 1 = Configuration Permitted
Inactivity Close Time	10	Unsigned16	2	Attribute value in seconds. It may be negotiated down by the responder.
Transmit Delay Time	12	Unsigned32	4	Attribute value in milliseconds. It may not be negotiated.

8.2.4.1.1.2 Response APDU Parameters

Same as Request APDU

8.2.4.1.1.3 Error APDU Parameters

The Common Error APDU is used, with the following specific parameter definitions.

The additional code parameter of the Common Error APDU carries the error severity, as defined below. The error severity is determined by the responding Type 5 FAL AE, and provides an indication of how severe the implementer of the Type 5 FAL AE considered the error condition to be. Its values are:

Value	Meaning
0	Not Severe
1	The error was caused by a normal inability to provide the requested service (e.g. parameter error or unsupported option)
2	The error was caused by a temporary resource condition.
3	The error was caused by a software or hardware fault that requires manual intervention or device reset

8.2.4.1.2 Idle (Unconfirmed Service Id = 3)

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

8.2.4.1.2.1 Request APDU Parameters

None.

8.2.4.2 SMK ASE Services

8.2.4.2.1 SM Find Tag Query (Unconfirmed Service Id = 1)

The table below describes the distribution that may be requested for this service by the SMK through the Type 5 FAL AE's SMK ASE. The Conveyed References shown in the table are shown in hex.

Table 504 – SMK Conveyed Reference Values

Conveyed Reference Meaning	Conveyed Reference		Destination Address	Distribution
	LLLL	NN.SS		
Config AP	00 00	00.00	Annunciation Multicast	Configuration Applications
Type 5 Device SMK	00 00	00.02	SMK Multicast	All Type 5 SMKs
			Individual	Individual Type 5 SMK
SMK Type 1 Local Link Group Address	00 00	*	SMK Multicast	All Type 5 SMKs and all Type 1 SMKs
			Individual	Individual LD SMK and all Type 1 SMKs accessible through the LD
SMK Type 1 Local Link Group Address	LL LL	*	SMK Multicast	All Type 5 SMKs and all SMKs on Type 1 link = LL LL
			Individual	Individual LD SMK and all SMKs on Type 1 link LL LL accessible through the linking device
Individual Type 1 Address	LL LL	NN.02	Individual	Individual Type 1 SMK accessible through the LD
NOTE * Specified in IEC 61158-6 as the Type 1 link specific address for "the SMAEs of all DLEs on the link"				

8.2.4.2.1.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Query Type	0	Unsigned8	1	Indicates the type of query: 0 = PD Tag query 1 = VFD tag query 2 = Function-Block tag query 3 = Element Id query 4 = PD Tag/VFD Reference query All other values are reserved.
Reserved	1	Octetstring	3	Reserved, set to 0
Element Id Or VFD Reference	4	Unsigned32	4	Service parameter. The Element Id is used for Query Type 3. The VFD Reference is used for Query Type 4.
PD Tag Or Object Tag	8	VisibleString	32	Service parameter
S-VFD Tag	40	VisibleString	32	Service parameter

8.2.4.2.2 SM Find Tag Reply (Unconfirmed Service Id = 2)

8.2.4.2.2.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Query Type	0	Unsigned8	1	Indicates the type of query: 0 = PD Tag query 1 = VFD tag query 2 = Function-Block tag query 3 = Element Id query 4 = PD Tag/VFD Reference query All other values are reserved.
Type 1 Node Address	1	Unsigned8	1	Type 1 Node Address portion of the service parameter 'Queried Object Type "X" Device Address'. Set to 0 if the Reply is for a Type 5 device
Type 1 Link Id	2	Unsigned16	2	Type 1 Link Id portion of the service parameter 'Queried Object Type "X" Device Address'. Set to 0 if the Reply is for a Type 5 device.
VFD Reference	4	Unsigned32	4	Service parameter
Queried Object Numeric Id	8	Unsigned32	4	Service parameter
Queried Object Address	12	Octetstring	16	Service parameter
Queried Object S-VFD Version Number	28	Unsigned32	4	Service parameter
Queried Object Device ID	32	VisibleString	32	Service parameter. Unused bytes in this field are set to ASCII space characters.
Reserved	64	Unsigned16	2	Reserved, set to 0.
Number in List of S-VFD Context Conveyed References	66	Unsigned16	2	This parameter indicates how many Conveyed Reference Selectors are contained in the List of Conveyed Reference Selectors parameter. This value of this attribute is 0 if the Query Type is "physical-device query".
List of S-VFD Context Conveyed References	68	Unsigned32	4x number in list	Service parameter. Each Selector is an Unsigned32 that immediately follows its predecessor in the series (no spaces or padding between them).

8.2.4.2.3 SM Identify (Confirmed Service Id = 3)

8.2.4.2.3.1 Request APDU Parameters

None.

8.2.4.2.3.2 Response APDU Parameters

Same as Address Assignment Request APDU Parameters.

8.2.4.2.4 SM Annunciate, Device Assignment, Clear Address (Confirmed/Unconfirmed Service Id = 8)

The request APDU body format defined below is used for the Annunciate, Device Assignment, and Clear Address service. It is also used as the response APDU body for the SM Identify service.

The sender of unconfirmed APDUs may use any of the APDU Trailer fields.

If the APDU is sent as a confirmed request, it always uses the Invoke Id in the APDU Trailer. If it is sent as a response, it always has the "Use" field set to identify the function requested, and it has the Invoke Id and the Conveyed Reference set to the value contained in the request. Responses are always sent to the individual address of the requester.

8.2.4.2.4.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Use	0	Unsigned8	1	Indicates the use of the APDU. The values of this field are: 0 = Unused 1 = Device Assignment 2 = Clear Address 3 = Annunciation 4 = SM Identify Response
Device Type	1	Unsigned8	1	This combines two service parameters, Device Type (bits 1-4) and Device Redundancy Capability (bits 7,8). A combination of bits 1-3 and 7,8 may be set for Type 5 devices. Bit 4 is used only for Type 1 devices. Non Type 5 devices participating in device assignment set the value of bits 1 through 6 of this parameter to 0. Bit 1 = Linking Device Bit 2 = Gateway Bit 3 = Type 5 Field Device Bit 4 = Type 1 Device Bit 5 = Reserved Bit 6 = Reserved Bit 7 = External Control Bit 8 = Automatic
Device Redundancy State	2	Unsigned16	2	SMK Attribute. Indicates the current device redundancy state of the device. The value 0 indicates that the device is not currently participating in device redundancy. If not 0, one and only one of bits 1, 2, and 3 must be set, and one and only one of bits 7 and 8 must be set: Bit 1 = Primary Bit 2 = Secondary Bit 3 = Member of Set Bit 4 = Reserved Bit 5 = Reserved Bit 6 = Reserved Bit 7 = External Control Bit 8 = Automatic
Device Index	4	Unsigned16	2	Service parameter 0 = Index not assigned 1 to Max Device Index = Device Index
Max Device Index	6	Unsigned16	2	Service parameter
SMK Multicast Address	8	Octetstring	16	Service parameter
Device Id	24	VisibleString	32	Service parameter. Unused bytes in this field are set to ASCII space characters.
Physical Device Tag	56	VisibleString	32	Service parameter. Unused bytes in this field are set to ASCII space characters.
Annunciation Repeat Time or New Type "X" Address	92	Unsigned32	4	Two service parameters share this field: Annunciation Repeat Time Used when the Device Type field indicates that the device is a Linking Device, Gateway, or Type 5 Field Device. Specifies the time in milliseconds.

				New Type "X" Address
				Used only to assign a new address to a Type 1 Device (Device Type field indicates that the device is an Type 1 Device). The Type 1 Device is to which the assignment is being made is identified in the Conveyed Reference of the APDU Header. The Conveyed Reference specifies the Type 1 Link Id (16 bits) and Node address (8 bits). The low order 8 bits are set to 0.
APDU Version Number	96	Unsigned32	4	Service parameter.
Device Version Number	100	Unsigned32	4	Service parameter.
Number of Entries in Version Number List	104	Unsigned32	4	This parameter indicates how many version numbers are contained in the Version Number List field below. Its value is 0 if the device is not a linking device or a gateway.
Version Number List	108	Unsigned32	4 x Number of Entries	Service parameter. This field is present only for linking devices and foreign I/O gateways. This field contains a list of numbers. Each is an Unsigned32 number. The number of entries in the list is specified by the field above "Number of Entries in Version Number List". For linking devices, each element in the list is constructed as follows: Bits 17-32: Type 1 Link Id Bits 9-16: Type 1 Node Id Bits 1-8: Version Number If the Conveyed Reference is 0, then this is a list of Type 1 live list version numbers, one for each Type 1 port attached to the linking device. In this case, the Node Id is unused and set to 0. The Type 1 Live List Version Number is defined in IEC 61158-4. The list is ordered by Type 1 bridge port number, with the Live List Version for the first port occurring first in the list. The value 0 is used for the Link Id bits if the port has not been assigned a link Id. If Conveyed Reference contains a Link Id, then this is a list of node address version numbers, one for each node id in the live list for the specified link. Node address version numbers begin at 0 and are incremented each time the node address is added to the live list or deleted from it. They roll over to 0 from 255. Node address version numbers are only used in responses to SMK Identify requests that identify an Type 1 link in the Conveyed Reference.

8.2.4.2.4.2 Response APDU Parameters

Same as Request APDU Parameters.

8.2.4.2.4.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3 Type 1 ASE Services

Unless specified otherwise, all parameters of the Type 1 Service APDUs and their values are defined in IEC 61158-5. NOTE that not all Type 1 services and their parameters are supported in this transfer syntax.

8.2.4.3.1 Reject

The Type 1 Reject Service is used only to indicate to the requester that the response is too long for Type 1 to convey it. Instead of defining a separate APDU here to convey this information, the Common Error APDU conveys it, using the Error Class "Reject" and the Reject Code as defined in the Type 1 abstract syntax for the Error Subclass.

8.2.4.3.2 Initiate (Confirmed Service Id = 96)

This service is used by a Type 5 client to open context to a Type 5 S-VFD or to a Type 1 AP. This APDU can be used to open a connection using the options shown in the following table to identify the target AP. The use of the Conveyed Reference for each of these options is defined in clause 8.2.2.1. They are:

Connect Option	Description
Context Id	This option allows the requesting application to provide the Context Id in the Conveyed Reference that is to be used to connect to the AP. If the AP is an S-VFD, then the Context Id is a Type 5 Server S-VFD Context Id obtained by configuration, by using the SMK Find Tag Query, or by other means. This Context Id may identify a generic context or a configured context. If the AP is a Type 1 AP, then this is a Linking Device port specific client Type 1 AREP numeric identifier with a configured Type 1 destination address.
MIB Access ¹	This option allows the requesting application to indicate that it desires to connect to a Type 1 or Type 5 MIB AP. The link id portion of the Conveyed Reference field indicates whether or not the MIB is in the Type 1 or Type 5 device.
NOTE 1The MIB AP can also be accessed using either of the Connect Options by providing the correct information.	

Client applications send Initiate Request APDUs to Type 5 FAL AEs to open S-VFD Contexts to Type 5 S-VFDs and to Type 1 APs. Type 1 AP access is provided by Linking Device Type 5 FAL AEs only. In both of these cases, the context used to access the server AP is identified by the Conveyed Reference field in the received request APDU Header. See clause 8.2.2.1 for the values that can be used.

To open an S-VFD Context, the Type 5 FAL AE locates the specified context endpoint and delivers the APDU to it. If it cannot locate the endpoint, it returns a negative response.

The context endpoint that receives the message may be an endpoint configured in the NMIB or it may be a generic (unconfigured) endpoint. If it is the generic endpoint, the generic endpoint creates a new endpoint with a new, unique context id and passes the request to it. The Type 1 AP Context state machine (see IEC 61158-6) is used by both configured and dynamically created endpoints to process the request message. Type 1 Initiate parameters that are not present in the Type 5 Initiate service are ignored (not used).

To open a context to a Type 1 AP, the Type 5 FAL AE locates the Type 1 stack identified by the link id portion of the Conveyed Reference and locates the Type 1 AREP specified in the lower 16 bits of the Conveyed Reference. If the Type 5 FAL AE is not able to locate the Type 1 AREP, or if it is not a client AREP (it must be a client AREP to forward requests onto the Type 1 link), the Type 5 FAL AE returns a negative response.

Once the Type 1 AREP has been located and validated, the Type 5 FAL AE creates a new Type 5 server context endpoint (Context Type = Type 1 CLIENT) that has a new, unique Context Id value and associates it with the Type 1 AREP. If the Type 1 AR is not already open, the Type 5 FAL AE opens it using the Type 1 Initiate service and any applicable parameters contained in the Type 5 Initiate Request APDU.

If the request APDU is received on a configuration session to establish a context to a local MIB AP (Type 5 or Type 1), or if there is already a context established to it, then the Type 5 FAL AE refuses the request and returns an Initiate Error APDU.

Once the context to the AP has been accepted, the Type 5 FAL AE returns the Initiate Response APDU. The Conveyed Reference field contained in this response APDU is the Context Id value for the newly created context endpoint. It is used for all subsequent APDUs sent through this context.

Once the Initiate Response APDU has been returned, the Type 5 FAL AE conveys request, response, and error APDUs between the context client and the Type 5 S-VFD or the Type 1 AP. However, if the context was not opened for configuration access to a Type 5 MIB S-VFD or Type 1 MIB AP, the Type 5 FAL AE issues a negative response if it receives an update request APDU from the client (an APDU for the Load Region ASE Download Services or the Variable ASE Write or Write with Subindex services). In this case, it does not convey the request to the S-VFD or AP.

8.2.4.3.2.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Connect Option	0	Unsigned8	1	This field indicates which option is to be used to open the S-VFD Context. Its values are: 1 = Context Id 2 = MIB Access
Access Protection Supported Calling/Called	8	Boolean	1	Service Parameter. Calling is used in the request and called in the response APDU.
Password and Access Groups Calling/Called	2	Unsigned16	2	Service Parameter. Calling is used in the request and called in the response APDU.
Version OD Calling	4	Integer16	2	Service Parameter. Calling is used in the request and called in the response APDU. This field is not used by the receiving Type 5 FAL AE in a linking device when initiating to a Type 1 AP.
AP Descriptor Calling	6	Unsigned16	2	Service Parameter.

8.2.4.3.2.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Version OD Called	0	Integer16	2	Service Parameter
AP Descriptor Called	2	OctetString	2	Service Parameter

8.2.4.3.2.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1. See Type 1 Initiate specific error codes.

8.2.4.3.3 Abort (Unconfirmed Service Id = 112)

8.2.4.3.3.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Abort Detail	0	OctetString	16	Supplied by the sender of the APDU.
Abort Identifier	16	Unsigned8	1	Service parameter, values defined in clause 4 above. .
Reason Code	17	Unsigned8	1	Service Parameter
Reserved	18	Unsigned16	2	Reserved, set to 0

8.2.4.3.4 Get Status (Confirmed Service Id = 0)

8.2.4.3.4.1 Request APDU Parameters

None

8.2.4.3.4.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Logical Status	0	Unsigned8	1	Service Parameter
Physical Status	1	Unsigned8	1	Service Parameter
Local Detail	2	OctetString	3	Service Parameter

8.2.4.3.5 Status Notification(Unconfirmed Service Id = 1)**8.2.4.3.5.1 Request APDU Parameters**

Same as Get Status Response APDU

8.2.4.3.6 Identify (Confirmed Service Id = 1)**8.2.4.3.6.1 Request APDU Parameters**

None

8.2.4.3.6.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Vendor Name	0	VisibleString	32	Service Parameter
Model Identifier	32	VisibleString	32	Service Parameter
Vendor Revision	64	VisibleString	32	Service Parameter

8.2.4.3.7 Get Attributes (Confirmed Service Id = 4)**8.2.4.3.7.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
All Attributes	0	Boolean	1	Service Parameter
Start Index Flag	1	Boolean	1	Service Parameter
Index	2	Unsigned32	4	Service Parameter

8.2.4.3.7.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
More Follows	0	Boolean	1	Service Parameter
Number of Object Descriptions	1	Unsigned8	1	Service Parameter
List of Object Descriptions	2	OctetString	N	Defined in clause 8.2.4.5

8.2.4.3.7.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.8 Begin Set Attributes (Confirmed Service Id = 28)

8.2.4.3.8.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Reserved	0	Unsigned16	2	Reserved, set to 0
Consequence	2	Unsigned16	2	Service Parameter. 0 = unused 1 = loading free of interaction 2 = reload, not free of interaction 3 = new load, not free of interaction

8.2.4.3.8.2 Response APDU Parameters

None

8.2.4.3.8.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.9 Set Attributes (Confirmed Service Id = 29)

8.2.4.3.9.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Number of Type 1 Managed AP Attributes	0	Unsigned8	1	Number of entries in List of Type 1 Managed AP Attributes
List of Type 1 Managed AP Attributes	1	OctetString	N	Defined in clause 8.2.4.5.

8.2.4.3.9.2 Response APDU Parameters

None

8.2.4.3.9.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.10 End Set Attributes (Confirmed Service Id = 30)

8.2.4.3.10.1 Request APDU Parameters

None

8.2.4.3.10.2 Response APDU Parameters

None

8.2.4.3.10.3 Error APDU Parameters

Defined by the OD Error APDU Parameters in clause 6.6.3.

8.2.4.3.11 Generic Initiate Download Sequence (Confirmed Service Id = 31)

This service corresponds to the Type 1 Initiate Load Service with the following parameter settings:

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

8.2.4.3.11.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.11.2 Response APDU Parameters

None

8.2.4.3.11.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.12 Generic Download Segment (Confirmed Service Id = 32)

This service corresponds to the Type 1 Push Segment Service.

8.2.4.3.12.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter
More Follows	4	Boolean	1	Service Parameter
Padding	5	OctetString	3	Set to 0.
Load Data	8	OctetString	N	Service Parameter

8.2.4.3.12.2 Response APDU Parameters

None

8.2.4.3.12.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.13 Generic Terminate Download Sequence (Confirmed Service Id = 33)

This service corresponds to the Type 1 Terminate Load Service.

8.2.4.3.13.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.13.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Terminate Reason	0	Boolean	1	Service Parameter

8.2.4.3.13.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.14 Initiate Download Sequence (Confirmed Service Id = 9)

This service corresponds to the Type 1 Initiate Load Service with the following parameter settings:

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

8.2.4.3.14.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.14.2 Response APDU Parameters

None

8.2.4.3.14.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.15 Download Segment (Confirmed Service Id = 10)

This service corresponds to the Type 1 Pull Segment Service.

8.2.4.3.15.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.15.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
More Follows	0	Boolean	1	Service Parameter
Load Data	1	OctetString	N	Service Parameter

8.2.4.3.15.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.16 Terminate Download Sequence (Confirmed Service Id = 11)

This service corresponds to the Type 1 Terminate Load Service.

8.2.4.3.16.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter
Reserved	4	Octetstring	3	Reserved, each byte is set to binary 0
Terminate Reason	7	Boolean	1	Service Parameter

8.2.4.3.16.2 Response APDU Parameters

None

8.2.4.3.16.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.17 Initiate Upload Sequence (Confirmed Service Id = 12)

This service corresponds to the Type 1 Initiate Load Service with the following parameter settings:

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

8.2.4.3.17.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.17.2 Response APDU Parameters

None

8.2.4.3.17.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.18 Upload Segment (Confirmed Service Id = 13)

This service corresponds to the Type 1 Pull Segment Service.

8.2.4.3.18.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.18.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
More Follows	0	Boolean	1	Service Parameter
Load Data	1	OctetString	N	Service Parameter

8.2.4.3.18.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.19 Terminate Upload Sequence (Confirmed Service Id = 14)

This service corresponds to the Type 1 Terminate Load Service.

8.2.4.3.19.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.19.2 Response APDU Parameters

None

8.2.4.3.19.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.20 Request Domain Download (Confirmed Service Id = 15)

This service corresponds to the Type 1 Initiate Load Service.

8.2.4.3.20.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter
Additional Information	2	VisibleString	N	Service Parameter

8.2.4.3.20.2 Response APDU Parameters

None

8.2.4.3.20.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.21 Request Domain Upload (Confirmed Service Id = 16)

This service corresponds to the Type 1 Initiate Load Service.

8.2.4.3.21.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Load Region Key Attribute	0	Unsigned32	4	Service Parameter
Additional Information	4	VisibleString	N	Service Parameter

8.2.4.3.21.2 Response APDU Parameters

None

8.2.4.3.21.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.22 Create Program Invocation (Confirmed Service Id = 17)

The Create Program Invocation corresponds to the Type 1 Create service.

8.2.4.3.22.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Reusable	0	Boolean	1	Attribute value
Reserved	1	Unsigned8	1	Reserved, set to 0
Number of Domain Indexes	2	Unsigned16	2	The number of domain indexes in this service
List Of Domain Indexes	4	Unsigned32	4 X Number of Domain Indexes	Attribute value

8.2.4.3.22.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Index	0	Unsigned32	4	Service Parameter

8.2.4.3.22.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.23 Delete Program Invocation (Confirmed Service Id = 18)

The Delete Program Invocation corresponds to the Type 1 Delete service.

8.2.4.3.23.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Index	0	Unsigned32	4	Service Parameter

8.2.4.3.23.2 Response APDU Parameters

None

8.2.4.3.23.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.24 Start (Confirmed Service Id = 19)**8.2.4.3.24.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter
Execution Argument	4	OctetString	N	Service Parameter

8.2.4.3.24.2 Response APDU Parameters

None

8.2.4.3.24.3 PI Error APDU Parameters

Common PI Error APDU

8.2.4.3.25 Stop (Confirmed Service Id = 20)**8.2.4.3.25.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.25.2 Response APDU Parameters

None

8.2.4.3.25.3 Error APDU Parameters

Common PI Error APDU

8.2.4.3.26 Resume (Confirmed Service Id = 21)**8.2.4.3.26.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter
Execution Argument	4	OctetString	N	Service Parameter

8.2.4.3.26.2 Response APDU Parameters

None

8.2.4.3.26.3 Error APDU Parameters

Common PI Error APDU

8.2.4.3.27 Reset (Confirmed Service Id = 22)**8.2.4.3.27.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.27.2 Response APDU Parameters

None

8.2.4.3.27.3 Error APDU Parameters

Common PI Error APDU

8.2.4.3.28 Kill (Confirmed Service Id = 23)

8.2.4.3.28.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.28.2 Response APDU Parameters

None

8.2.4.3.28.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.29 Read (Confirmed Service Id = 2)

8.2.4.3.29.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.29.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Value	0	OctetString	N	Service Parameter

8.2.4.3.29.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.30 Read with Subindex (Confirmed Service Id = 82)

This service corresponds to the Type 1 Read service.

8.2.4.3.30.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter
Component ID	4	Unsigned32	4	Service Parameter

8.2.4.3.30.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Value	0	OctetString	N	Service Parameter

8.2.4.3.30.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.31 Write (Confirmed Service Id = 3)

8.2.4.3.31.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter
Value	4	OctetString	N	Service Parameter

8.2.4.3.31.2 Response APDU Parameters

None

8.2.4.3.31.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.32 Write with Subindex (Confirmed Service Id = 83)

This service corresponds to the Type 1 Write service.

8.2.4.3.32.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter
Component ID	4	Unsigned32	4	Service Parameter
Value	8	OctetString	N	Service Parameter

8.2.4.3.32.2 Response APDU Parameters

None

8.2.4.3.32.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.33 Define Variable List (Confirmed Service Id = 7)

This service corresponds to the Type 1 Create service.

8.2.4.3.33.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Number of Indexes	0	Unsigned32	4	Number of Variable Indexes in the list that follows.
List Of values	4	Unsigned32	4 x Number of Indexes	Attribute value

8.2.4.3.33.2 Response APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Numeric Id	0	Unsigned32	4	Service Parameter

8.2.4.3.33.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.34 Delete Variable List (Confirmed Service Id = 8)

This service corresponds to the Type 1 Delete service.

8.2.4.3.34.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned32	4	Service Parameter

8.2.4.3.34.2 Response APDU Parameters

None

8.2.4.3.34.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.35 Information Report (Unconfirmed Service Id = 0)

8.2.4.3.35.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned16	2	Service Parameter
Value	2	OctetString	N	Service Parameter

8.2.4.3.36 Information Report with Subindex (Unconfirmed Service Id = 16)

This service corresponds on the Type 1 Information Report service.

8.2.4.3.36.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned16	2	Service Parameter
Component ID	2	Unsigned16	2	Service Parameter
Value	4	OctetString	N	Service Parameter

8.2.4.3.37 Information Report On Change (Unconfirmed Service Id = 17)

This service corresponds to the Type 1 Information Report service.

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

8.2.4.3.37.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned16	2	Service Parameter
Value	2	OctetString	N	Service Parameter

8.2.4.3.38 Information Report On Change with Subindex (Unconfirmed Service Id = 18)

This service corresponds to the Type 1 Information Report service.

This APDU is used to send data that is published only when its value has changed. The Type 5 S-VFD Context that controls the publishing indicates if this service is to be used.

8.2.4.3.38.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Key Attribute	0	Unsigned16	2	Service Parameter
Component ID	2	Unsigned16	2	Service Parameter
Value	4	OctetString	N	Service Parameter

8.2.4.3.39 Event Notification (Unconfirmed Service Id = 2)

8.2.4.3.39.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Index	0	Unsigned32	4	Service Parameter
Event Number	4	Unsigned32	4	Service Parameter
Data	8	OctetString	N	Service Parameter

8.2.4.3.40 Alter Event Condition Monitoring (Confirmed Service Id = 24)**8.2.4.3.40.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Index	0	Unsigned32	4	Service Parameter
Reserved	4	Octetstring	3	Reserved, each byte is set to binary 0
Enabled	7	Boolean	1	Service Parameter

8.2.4.3.40.2 Response APDU Parameters

None

8.2.4.3.40.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.3.41 Acknowledge Event Notification (Confirmed Service Id = 25)**8.2.4.3.41.1 Request APDU Parameters**

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Index	0	Unsigned32	4	Service Parameter
Event Number	4	Unsigned32	4	Service Parameter
Data	8	OctetString	N	Service Parameter

8.2.4.3.41.2 Response APDU Parameters

None

8.2.4.3.41.3 Error APDU Parameters

Defined by the Common Error APDU Parameters in clause 8.5.1.

8.2.4.4 LAN Redundancy Services

8.2.4.4.1 LAN Redundancy Diagnostic Message (Unconfirmed Service Id = 1)

8.2.4.4.1.1 Request APDU Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Device Index	0	Unsigned16	2	Service parameter 0 = Index not assigned 1 – 65535 = Device Index
Number of Interfaces	2	Unsigned8	1	Service parameter.
Type 5 Interface used for Transmission of this Diagnostic APDU	3	Unsigned8	1	Type 5 Interface used to transmit this Diagnostic APDU. 0 = Interface A 1 = Interface B
Diagnostic Message Interval	4	Unsigned 32	4	Service parameter.
Max Device Index	8	Unsigned16	2	Service parameter.
Reserved	10	Unsigned16	2	Reserved. Set to zero
Array of Interface A Statuses	12	Array of Unsigned32	(4*Max Device Index)/32	This field represents List of Network Interface Status parameter for Interface A. Each bit in this array* represents a remote device that sends diagnostic APDUs to the device that is sending this APDU (the reporting device). The following values are used for each bit: 0 = Diagnostic APDUs successfully received from device N on this interface of this device 1 = Diagnostic APDUs not received on this interface of this device
Array of Interface B Statuses	12 + 4 * (Max. Device Index/32)	Array of Unsigned32	(4*Max Device Index)/32	This field represents List of Network Interface Status parameter for Interface B. Its definition is the same as the definition for Array of Interface A Statuses.
Array of crossed cable status	12 + 8 * (Max. Device Index/32)	Array of Unsigned32	(4*Max Device Index)/32	This field represents List of Crossed Cable Status parameter. Each bit in this array* represents a device that sends diagnostic APDUs to the device that is sending this APDU (the reporting device). Each remote device's bit represents whether the interface it is using to send diagnostic APDUs, as indicated by the <i>Interface Used for Transmission of this Diagnostic APDU</i> field, is the same interface used by the reporting device to receive the APDUs. The following values are used for each bit: 0 = Cables not crossed (interfaces are different) or <i>Crossed Cable Detection Enabled</i> is <i>false</i> . 1 = Cables are crossed (interfaces are different)
* The bit that identifies a device is located by its bit position in the array. Dividing the device's Device Index by 32 yields the index of the zero-based Unsigned32 array element. The remainder from the division identifies the specific bit in the array element. Bit 1 in an Unsigned32 is the most significant bit. Therefore, a device with Device Index 33 would be represented by the most significant bit of the element of the array accessed with zero-based index = 2.				

8.2.4.5 List of Type 1 Managed AP Attributes

This section defines the Type 1 Managed AP attributes conveyed.

The high order byte indicates, when 1, that the long form is follows, and when 0, that the short form follows. The low order byte contains the Class Id value. Class Ids are defined in IEC 61158-5, clause 7.5, except as noted below.

8.2.4.5.1 Structure of the List of Managed AP Attributes

In the table below, the Class Id field has been added to provide for consistent decoding of the List of Managed AP Attributes returned in the Get Attribute response APDU. The value for this Class Id is 11.

Field Name	Octet Offset	Data Type	Octet Length
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
ROM/RAM Flag	6	Unsigned16	2
Max Name Length	8	Unsigned32	4
Access Protection Supported	12	Unsigned32	4
Version Of List	16	Unsigned32	4
Number Of Objects (Data Types)	20	Unsigned32	4
Static Object Numeric Identifier	24	Unsigned32	4
Number Of Static Objects	28	Unsigned32	4
Variable List Numeric Identifier	32	Unsigned32	4
Number Of Variable List Objects	36	Unsigned32	4
Function Invocation Numeric Identifier	40	Unsigned32	4
Number Of Function Invocation Objects	44	Unsigned32	4
* List Local Reference	48	Unsigned64	8
* Data Type Local Reference	56	Unsigned64	8
* Static Object Local Reference	64	Unsigned64	8
* Variable List Local Reference	72	Unsigned64	8
* Function Invocation Local Reference	80	Unsigned64	8
* If All Attributes are not requested, these fields are not present			

8.2.4.5.2 Structure of a Load Region

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Local Region State	6	Unsigned8	1
Upload State	7	Unsigned8	1
Load Region Size	8	Unsigned32	4
Number of Related Objects In Use	12	Unsigned32	4
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
Long Form Add'l Fields*			
Local Detail	0 + Long Form Offset	Unsigned64	8
Password	8 + Long Form Offset	Unsigned8	1
Access Groups	9 + Long Form Offset	Unsigned8	1
Access Rights	10 + Long Form Offset	Unsigned16	2
Load Image Name	12 + Long Form Offset	VisibleString	**Max Name Length
* If the Short Form is requested, these fields are not present.			
** If Max Name Length in the List of Managed AP Attributes is 0, then this field is not present			

8.2.4.5.3 Structure of a Function Invocation

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Deletable	6	Unsigned8	1
Reusable	7	Unsigned8	1
Number Of Related Objects in Use	8	Unsigned32	4
Function invocation State	12	Unsigned8	1
Reserved	13	Octetstring	3
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
List of Related Objects Numeric Identifiers	20+ Extension Length	Unsigned32	* Number Of Related Objects x 4
Long Form Add'l Fields**	20+ Extension Length + (Number Of Domains x 4)		
Password	0 + Long Form Offset	Unsigned8	1
Access Groups	1 + Long Form Offset	Unsigned8	1
Access Rights	2 + Long Form Offset	Unsigned16	2
Function Innvocation Name	4 + Long Form Offset	VisibleString	*** Max Name Length
* If Number Of Domains is 0, then this field is not present			
** If the Short Form is requested, these fields are not present.			
*** If Max Name Length in the List of AP Managed Attributes is 0, then this field is not present			

8.2.4.5.4 Structure of a Simple Variable

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Reserved	6	Unsigned16	2
Data Type Numeric Identifier	8	Unsigned32	4
Octet Length	12	Unsigned32	4
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
Long Form Add'l Fields*	20+ Extension Length		
Local Detail	0 + Long Form Offset	Unsigned64	8
Password	8 + Long Form Offset	Unsigned8	1
Access Groups	9 + Long Form Offset	Unsigned8	1
Access Rights	10 + Long Form Offset	Unsigned16	2
Variable Name	12 + Long Form Offset	VisibleString	**Max Name Length
* If the Short Form is requested, these fields are not present.			
** If Max Name Length in the List of AP Managed Attributes is 0, then this field is not present			

8.2.4.5.5 Structure of an Array

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Number of Elements	6	Unsigned16	2
Data Type Numeric Identifier	8	Unsigned32	4
Element Octet Length	12	Unsigned32	4
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
Long Form Add'l Fields*			
Local Detail	0 + Long Form Offset	Unsigned64	8
Password	8 + Long Form Offset	Unsigned8	1
Access Groups	9 + Long Form Offset	Unsigned8	1
Access Rights	10 + Long Form Offset	Unsigned16	2
Variable Name	12 + Long Form Offset	VisibleString	**Max Name Length
* If the Short Form is requested, these fields are not present.			
** If Max Name Length in the List of Managed AP Attributes is 0, then this field is not present			

8.2.4.5.6 Structure of a Record

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Reserved	6	Unsigned16	2
Data Type Numeric Identifier	8	Unsigned32	4
Reserved	12	Unsigned32	4
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
Long Form Add'l Fields*			
List of Local Detail	0 + Long Form Offset	Unsigned64	**K x 8
Password	(K x 8) + Long Form Offset	Unsigned8	1
Access Groups	1 + (K x 8) + Long Form Offset	Unsigned8	1
Access Rights	2 + (K x 8) + Long Form Offset	Unsigned16	2
Record Name	4 + (K x 8) + Long Form Offset	VisibleString	***Max Name Length
* If the Short Form is requested, these fields are not present.			
** K represents the number of fields in the record.			
*** If Max Name Length in the List of Managed AP Attributes is 0, then this field is not present			

8.2.4.5.7 Structure of a Variable List

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Number of Entries	6	Unsigned16	2
Data Type Numeric Identifier	8	Unsigned32	4
Deletable	12	Unsigned8	1
Reserved	13	OctetString	3
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
Long Form Add'l Fields*			
List of Variables	20+ Extension Length		
List of Variables	0 + Long Form Offset	Unsigned32	4 x Number of Elements
Password	(**K x 4) + Long Form Offset	Unsigned8	1
Access Groups	1 + (K x 4) + Long Form Offset	Unsigned8	1
Access Rights	2 + (K x 4) + Long Form Offset	Unsigned16	2
Variable List Name	4 + (K x 4) + Long Form Offset	VisibleString	***Max Name Length
* If the Short Form is requested, these fields are not present.			
** K represents the Number of Elements in the record.			
*** If Max Name Length in the List of Managed AP Attributes is 0, then this field is not present			

8.2.4.5.8 Structure of a Data Type

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Octet Length	6	Unsigned8	1
Symbolic Name Length	7	Unsigned8	1
*Symbolic Name	8	VisibleString	Symbolic Name Length
* If the Symbolic Name Length is 0, then this field is not present			

8.2.4.5.9 Structure of a Data Type Structure Description

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Number of Fields	6	Unsigned16	2
List of Fields	8	Structure	8 x Number of Elements
Field Data Type		Unsigned32	4
Length		Unsigned32	4
Symbolic Name Length	7	Unsigned8	1
*Symbolic Name	8	VisibleString	Symbolic Name Length
* If the Symbolic Name Length is 0, then this field is not present			

8.2.4.5.10 Structure of an Event

Field Name	Octet Offset	Data Type	Octet Length
Short Form Fields			
Numeric Identifier	0	Unsigned32	4
Class Id	4	Integer16	2
Enabled	6	Unsigned16	2
Event Data Variable ID Specifier	8	Unsigned32	4
Event Data Length	12	Unsigned32	4
Extension Length	16	Unsigned32	4
Extension Data	20	OctetString	Extension Length
Long Form Add'l Fields*			
Password	20 + Extension Length		
Access Groups	0 + Long Form Offset	Unsigned8	1
Access Rights	1 + Long Form Offset	Unsigned8	1
Event Name	2 + Long Form Offset	Unsigned16	2
	4 + Long Form Offset	VisibleString	**Max Name Length
* If the Short Form is requested, these fields are not present.			
** If Max Name Length in the List of Managed AP Attributes is 0, then this field is not present			

8.2.5 Error APDU Bodies

8.2.5.1 Common Error Parameters

The following error parameter format is used for many services.

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Error Class	0	Unsigned8	1	This parameter identifies the category of the error. Its values are defined in the Error Class production of the Type 1 Abstract Syntax. The values are the tagged values. The value 9 has been added for the Type 5 Initiate Error APDU use.
Error Subclass	1	Unsigned8	1	This parameter identifies the specific type of error within the class. Its values are defined in the Error Class production of the Type 1 Abstract Syntax. The values are the enumerated values with parenthesized values, such as other = 0.
Reserved	2	Unsigned8	1	Reserved set to 0
Additional Code	3	Unsigned8	1	This parameter provides an additional code that is associated with the error.
Additional Description	4	VisibleString	16	This parameter provides 16 octets of additional information that is associated with the error.

8.2.5.2 FI Error Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Pi State	0	Unsigned8	1	Defined in Service Parameter.
Error Class	1	Unsigned8	1	Defined in Service Parameter.
Error Subclass	2	Unsigned8	1	Defined in Service Parameter.
Additional Code	3	Unsigned8	1	Defined in Service Parameter.
Additional Description	4	VisibleString	16	Defined in Service Parameter.

8.2.5.3 OD Error Parameters

Parameter Name	Octet Offset	Data Type	Octet Length	Description
Index	0	Unsigned32	4	Defined in the Service Parameter.
Reserved	4	Unsigned8	1	Reserved, set to 0
Error Class	5	Unsigned8	1	Defined in the Service Parameter.
Error Subclass	6	Unsigned8	1	Defined in the Service Parameter.
Additional Code	7	Unsigned8	1	Defined in the Service Parameter.
Additional Description	8	VisibleString	16	Defined in the Service Parameter.

8.3 FAL Protocol State Machine Structure

The FAL Protocol state machine structure defined in clause 4.3 applies.

8.4 AP Context State Machine

The AP context machines defined in clause 4.4 is used for the S-VFD context with the following exceptions. The Type 5 Initiate service replaces the Type 1 Initiate service. Type 1 AP attributes that are not defined as S-VFD Context attributes and Type 1 Initiate service parameters that are not defined for the S-VFD ASE Initiate service are ignored in the state machine.

8.5 FAL Service Protocol Machine (FSPM)

The Type 1 FSPM is used with the following modifications:

- Service primitives between the FSPM and the ARPM contain parameters as defined in clauses 8.6.1.1.1 and 8.6.2.1.1.
- Service primitives in the Type 5 FSPM contain the svfdContext_id parameter that is the Numeric Id of the S-VFD Context.

8.6 Application Relationship Protocol Machines (ARPMs)

8.6.1 Client / Server Session ARPM

8.6.1.1 Primitive Definitions

8.6.1.1.1 Primitives Exchanged between ARPM and FSPM

Table 505 – Primitives issued by FSPM to ARPM

Primitive Name	Source	Associated parameters	Functions
EST_req	FSPM	user_data, remote_address	This is an FAL internal primitive used to convey an Establish request primitive from the FSPM to the ARPM.
EST_rsp(+)	FSPM	user_data, svfdContext_id, service_type	This is an FAL internal primitive used to convey an Establish response(+) primitive from the FSPM to the ARPM.
EST_rsp(-)	FSPM	user_data, svfdContext_id, service_type	This is an FAL internal primitive used to convey an Establish response(-) primitive from the FSPM to the ARPM.
Abort_req	FSPM	identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort request primitive from the FSPM to the ARPM.
CS_req	FSPM	user_data, svfdContext_id, service_type	This is an FAL internal primitive used to convey a Confirmed Send (CS) request primitive from the FSPM to the ARPM.
CS_rsp	FSPM	user_data, svfdContext_id, service_type	This is an FAL internal primitive used to convey a Confirmed Send (CS) response primitive from the FSPM to the ARPM.
UCS_req	FSPM	user_data, svfdContext_id, service_type, remote_address	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) request primitive from the FSPM to the ARPM.

Table 506 – Primitives issued by ARPM to FSPM

Primitive Name	Source	Associated Parameters	Functions
EST_ind	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish indication primitive from the ARPM to the FSPM.
EST_cnf(+)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish confirmation(+) primitive from the ARPM to the FSPM.
EST_cnf(-)	ARPM	arep_id, user_data	This is an FAL internal primitive used to convey an Establish confirmation(-) primitive from the ARPM to the FSPM.
Abort_ind	ARPM	arep_id, locally_generated, identifier, reason_code, additional_detail	This is an FAL internal primitive used to convey an Abort primitive from the ARPM to the FSPM.
CS_ind	ARPM	arep_id, svfdContext_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) indication primitive from the ARPM to the FSPM.
CS_cnf(+)	ARPM	arep_id, , svfdContext_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation(+) primitive from the ARPM to the FSPM.
CS_cnf(-)	ARPM	arep_id, svfdContext_id, service_type, user_data	This is an FAL internal primitive used to convey a Confirmed Send (CS) confirmation(-) primitive from the ARPM to the FSPM.
UCS_ind	ARPM	arep_id, svfdContext_id, service_type, user_data	This is an FAL internal primitive used to convey an Unconfirmed Send (UCS) indication primitive from the ARPM to the FSPM.

8.6.1.1.2 Parameters of FSPM/ARPM Primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in the table below.

Table 507 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter Name	Description
arep_id	This parameter is used to unambiguously identify an instance of the AREP that has issued a primitive. A means for such identification is not specified by this standard.
user_data	This parameter conveys FAL-User data.
locally_generated	This parameter conveys value that is used for the Locally_Generated parameter.
identifier	This parameter conveys value that is used for the Identifier parameter.
reason_code	This parameter conveys value that is used for the Reason_Code parameter.
additional_detail	This parameter conveys value that is used for the Additional_Detail parameter.
remote_address	This parameter conveys value of the remote dlcep address.
svfdContext_id	This parameter is used to unambiguously identify an instance of the application that has issued a primitive or that is the primitive destination. A means for such identification is not specified by this standard.
service_type	This parameter conveys the protocol Id and the service within the protocol.

8.6.1.2 DLL Mapping of Client / Server AREP Class

This clause describes the mapping of the Client / Server AREP Class to the Socket model. A means to configure and monitor the values of these attributes will be provided in the future IEC 61158-7.

The DLL Mapping attributes and their permitted values used with the Client / Server AREP class are defined in this clause.

CLASS: **ClientServer**

PARENT CLASS: **AR Endpoint**

ATTRIBUTES:

1	(m)	KeyAttribute:	LocalAndRemoteAddresses
1.1	(m)	KeyAttribute:	LocalAddress
1.2	(m)	KeyAttribute:	RemoteAddress
2	(m)	Attribute:	TransferType (CONN, CNLS)
3	(m)	Attribute:	BufferSize
4	(m)	Attribute:	MaxTransmitDelayTime
5	(m)	Attribute:	InactivityCloseTime
6	(m)	Attribute:	HdrOptions
7	(m)	Attribute:	GuardBand

8.6.1.2.1.1 Attributes

LocalAndRemoteAddresses

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

TransferType

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

BufferSize

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

MaxTransmitDelayTime

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

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

InactivityCloseTime

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

HdrOptions

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

TransferType

This attribute specifies the transfer protocol used for the session. Its values are CONN and CNLS.

GuardBand

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

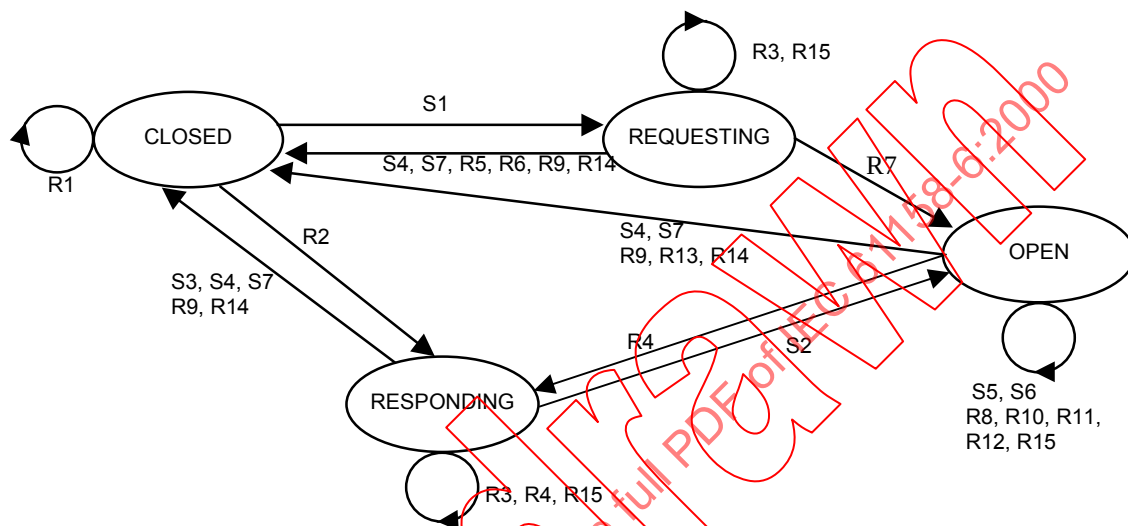
8.6.1.3 Client / Server AREP State Machine

8.6.1.3.1 Client / Server ARPM States

The defined states and their descriptions of the Client / Server ARPM are listed below. For CONN, the closed state is never left until the connection has been established. How this connection is established is implementation dependent:

Table 508 – Client / Server ARPM states

States	Description
CLOSED	The AREP is defined, but not capable of sending or receiving arbitrary FAL-PDUs. It may send or receive Establish service FAL-PDUs while in this state.
OPEN	The AREP is defined and capable of sending or receiving FAL-PDUs.
REQUESTING	The AREP has sent an Establish Request FAL-PDU and is waiting for a response from the remote AREP.
RESPONDING	The AREP has received an Establish Request FAL-PDU, delivered an Establish.ind primitive and is waiting for a response from its user.
SAME	Indicates that the next state is the same as the current state.

**Figure 146 – State Transition Diagram of Client / Server ARPM**

8.6.1.3.2 Client / Server ARPM state table

Table 509 – Client / Server ARPM state table - Sender transactions

#	Current State	Event Action	Next State
S1	CLOSED	EST_req && Role = CLIENT <pre> FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildFAL-PDU (fal_service_type := service_type, confirmed_unconfirmed := "Confirmed", fal_data := user_data) } </pre> * Not used for TransferType = CONN	REQUESTING
S2	RESPONDING	EST_rsp(+) && Role = SERVER <pre> FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildFAL-PDU (fal_service_type := service_type, confirmed_unconfirmed := "Confirmed", fal_data := user_data) } </pre> * Not used for TransferType = CONN	OPEN
S3	RESPONDING	EST_rsp(-) && Role = SERVER <pre> FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildFAL-PDU (fal_service_type := service_type, confirmed_unconfirmed := "Confirmed", fal_data := user_data) } </pre> * Not used for TransferType = CONN	CLOSED
S4	RESPONDING REQUESTING OPEN	Abort_req	CLOSED
S5	OPEN	UCS_req <pre> FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildFAL-PDU (fal_service_type := service_type, confirmed_unconfirmed := "Unconfirmed", fal_data := user_data) } </pre> * Not used for TransferType = CONN	OPEN

#	Current State	Event Action	Next State
S6	OPEN	CS_req CS_rsp FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildFAL-PDU (fal_service_type := service_type, confirmed_unconfirmed := "Confirmed", fal_data := user_data) } * Not used for TransferType = CONN	OPEN
S7	REQUESTING RESPONDING OPEN	"Inactivity Time Out" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := " AR Inactivity" }	CLOSED

Table 510 – Client / Server ARPM state table - Receiver transactions

#	Current State	Event Action	Next State
R1	CLOSED	FAL-PDU_ind && Role = "SERVER" && FAL_Pdu_SvcType (fal_pdu) = "EST_ReqPDU" && ConfigurationAREPisOK(fal_pdu) = False <pre> FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildErrorResponseFAL-PDU (fal_data = fal_pdu, Error_Class = "Service", Error_Subclass = "object-constraint-conflict") } </pre> * Not used for TransferType = CONN	CLOSED
R2	CLOSED	FAL-PDU_ind && Role = "SERVER" && FAL_Pdu_SvcType (fal_pdu) = "EST_ReqPDU" && ConfigurationAREPisOK(fal_pdu) = True RemoteAddress = GetRemoteAddress()* <pre> EST_ind { arep_id := GetArepld (), user_data := GetUserData (fal_pdu) } </pre> * Not used if TransferType = CONN	RESPOND- ING
R3	REQUEST- ING RESPOND- ING	FAL-PDU_ind &&FAL_Pdu_SvcType (fal_pdu) = "CS_ReqPDU" <pre> FAL-PDU_req { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemoteAddress*, fal_pdu := BuildErrorResponseFAL-PDU (fal_data = fal_pdu, Error_Class = "Service", Error_Subclass = "object-statet-conflict") } </pre> * Not used for TransferType = CONN	SAME
R4	OPEN RESPOND- ING	FAL-PDU_ind && Role = "SERVER" && FAL_Pdu_SvcType (fal_pdu) = "EST_ReqPDU" <pre> Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := " Open session request on open(ing) session " } EST_ind { arep_id := GetArepld (), user_data := GetUserData(fal_pdu) } </pre>	RESPOND- ING

#	Current State	Event Action	Next State
R5	REQUEST- ING	FAL-PDU_ind && Role = "CLIENT" && FAL_Pdu_SvcType (fal_pdu) = "EST_ErrPDU" && InvokeldExistent(GetArepld(), GetInvokeld(fal_pdu)) EST_cnf(-) { arep_id := GetArepld (), user_data := GetUserData(fal_pdu) }	CLOSED
R6	REQUEST- ING	FAL-PDU_ind && Role = "CLIENT" && FAL_Pdu_SvcType (fal_pdu) = "EST_RspPDU" && NegotiatedParametersOk(fal_pdu) = False && InvokeldExistent(GetArepld(), GetInvokeld(fal_pdu)) Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Negotiation Failure for Parameter 'X'" }	CLOSED
R7	REQUEST- ING	FAL-PDU_ind && Role = "CLIENT" && FAL_Pdu_SvcType (fal_pdu) = "EST_RspPDU" && NegotiatedParametersOk = True && InvokeldExistent(GetArepld(), GetInvokeld(fal_pdu)) BufferSize := GetBufferSize(fal_pdu) InactivityCloseTime := GetInactivityCloseTime(fal_pdu) MsgHdrOptions = GetMsgHdrOptions(fal_pdu) EST_cnf(+) { arep_id := GetArepld (), user_data := GetUserData(fal_pdu) }	OPEN
R8	OPEN	FAL-PDU_ind && Role = "CLIENT" && FAL_Pdu_SvcType (fal_pdu) = "EST_RspPDU" && NegotiatedParametersOk(fal_pdu) = True && InvokeldExistent(GetArepld(), GetInvokeld(fal_pdu)) BufferSize := GetBufferSize(fal_pdu) InactivityCloseTime := GetInactivityCloseTime(fal_pdu) MsgHdrOptions = GetMsgHdrOptions(fal_pdu) EST_cnf(+) { arep_id := GetArepld (), svfdContext_id := GetSvfdContextId (), user_data := GetUserData(fal_pdu) }	OPEN

#	Current State	Event Action	Next State
R9	OPEN REQUEST- ING RESPOND- ING	FAL-PDU_ind && MessageValid(fal_pdu) = False FAL-PDU_req* { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemotelpAddress**, fal_pdu := BuildErrorResponseFAL-PDU (fal_data = fal_pdu, Error_Class = "Service", Error_Subclass = "Parameter inconsistent") } Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := " Invalid message " } * Not used for CS_RspPDU or CS_ErrPDU ** Not used for TransferType = CONN	CLOSED
R10	OPEN	FAL-PDU_ind && FAL_Pdu_SvcType (fal_pdu) == "UCS_ReqPDU" && MsgHdrOptions == GetMsgHdrOptions(fal_pdu) UCS_ind { arep_id := GetArepld (), svfdContext_id := GetSvfdContextId (), service_type := GetServiceType(fal_pdu), user_data := GetUserData(fal_pdu) }	OPEN
R11	OPEN	FAL-PDU_ind && FAL_Pdu_SvcType (fal_pdu) == "CS_ReqPDU" && MsgHdrOptions == GetMsgHdrOptions(fal_pdu) CS_ind { arep_id := GetArepld (), svfdContext_id := GetSvfdContextId (), service_type := GetServiceType(fal_pdu), user_data := GetUserData(fal_pdu) }	OPEN
R12	OPEN	FAL-PDU_ind && (FAL_Pdu_SvcType (fal_pdu) == "CS_RspPDU" FAL_Pdu_SvcType (fal_pdu) == "CS_ErrPDU") && MsgHdrOptions == GetMsgHdrOptions(fal_pdu) && InvokeldExistent(GetArepld(), GetInvokeld(fal_pdu)) CS_cnf { arep_id := GetArepld (), svfdContext_id := GetSvfdContextId (), service_type := GetServiceType(fal_pdu), user_data := GetUserData(fal_pdu) }	OPEN

#	Current State	Event Action	Next State
R13	OPEN	<pre> FAL-PDU_ind && (FAL_Pdu_SvcType (fal_pdu) == "UCS_ReqPDU" FAL_Pdu_SvcType (fal_pdu) == "CS_ReqPDU" FAL_Pdu_SvcType (fal_pdu) == "CS_RspPDU" FAL_Pdu_SvcType (fal_pdu) == "CS_ErrPDU") && MsgHdrOptions <> GetMsgHdrOptions(fal_pdu) FAL-PDU_req* { smpm_service_name := "SMPM_Send", arep_id := GetArepld (), remote_address := RemotelpAddress**, fal_pdu := BuildErrorResponseFAL-PDU (fal_data = fal_pdu, Error_Class = "Service", Error_Subclass = "Parameter inconsistent") } Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Trailer fields present on message is different than negotiated " } </pre> * Not used for CS_RspPDU or CS_Err_PDU ** Not used for TransferType = CONN	CLOSED
R14	OPEN REQUEST- ING RESPOND- ING	<pre> ErrorToARPM && ErrorType = "socket disconnect" Abort_ind { arep_id := GetArepld (), locally_generated := "True", identifier := "FAL", reason_code := "Socket disconnected" } </pre>	CLOSED
R15	OPEN REQUEST- ING RESPOND- ING	<pre> ErrorToARPM (No actions taken. See NOTE below.) </pre> NOTE It is a local matter to report this error status to network management entities. The ARPM does not abort the existing connections. The FAL user may issue an Abort request to disconnect the current connection, depending on the type of status information conveyed by the ErrorToARPM primitive.	SAME

8.6.1.3.3 Functions used by Client / Server ARPM

Table 511 – Function GetArepld

Name	GetArepld ()	Used in	ARPM
Input		Output	
(none)		AREP Identifier	
Function			
Returns a value that can unambiguously identify the current AREP.			

Table 512 – Function BuildFAL-PDU

Name	BuildFAL-PDU	Used in	ARPM
Input		Output	
fal_service_type, confirmed_unconfirmed, fal_data		fal_pdu	
Function	Builds an FAL-PDU out of the parameters given as input variables. This includes: - Building the FAL pdu header - adding the fal_data (AR user data) - Generating the FALpdu trailer, including Message Number and Invoke ID Description of parameters: fal_service_type This parameter conveys the protocol Id and the service within the protocol confirmed_unconfirmed Enumeration type (Confirmed, Unconfirmed) fal_data AR user data		

Table 513 – Function FAL_Pdu_SvcType

Name	FAL_Pdu_Type	Used in	ARPM
Input		Output	
dls_user_data	fal_pdu	One of the FAL-PDU types defined in clause 4.	
Function			
This function decodes the FAL-PDU that is conveyed in the fal_pdu parameter and retrieves one of the FAL-PDU service types. The FAL-PDU service types are: EST_ReqPDU EST_RspPDU EST_ErrPDU CS_ReqPDU CS_RspPDU CS_ErrPDU UCS_ReqPDU			

Table 514 – Function GetRemoteAddress

Name	GetRemoteAddress ()	Used in	ARPM
Input		Output	
-		address of remote endpoint	
Function	Returns the address of the sender of a received message.		

Table 515 – Function GetSvfdContextId

Name	GetSvfdContextId ()	Used in	ARPM
Input		Output	
(none)		Svfd context Identifier	
Function	Returns a value that can unambiguously identify an instance of the application context.		

Table 516 – Function ConfigurationAREPisOK

Name	ConfigurationAREPs ()	Used in	ARPM
Input		Output	
fal_pdu		True False	
Function	Returns True if: - The request fal_pdu is not a configuration request - The request fal_pdu is a configuration request and there are no other configuration ARs established Returns False if: - The request fal_pdu is a configuration request and there is already another configuration AR established.		