

INTERNATIONAL STANDARD

IEC
61375-1

First edition
1999-09

**Electric railway equipment –
Train bus –**

Part 1: Train Communication Network

Matériel électrique ferroviaire – Bus de train –

Partie 1: Réseau embarqué de train



Reference number
IEC 61375-1:1999(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
61375-1

First edition
1999-09

Electric railway equipment – Train bus –

Part 1: Train Communication Network

Matériel électrique ferroviaire – Bus de train

Partie 1: Réseau embarqué de train

© IEC 1999 – 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

XH

For price, see current catalogue

CONTENTS

	Page
FOREWORD	27
INTRODUCTION	29
Clause	
1 General	31
1.1 Scope	31
1.2 Normative references	31
1.3 Terms and definitions	33
1.4 Legends and abbreviations	51
1.5 Conventions	53
1.5.1 Base of Numeric Values	53
1.5.2 Naming conventions	53
1.5.3 Time naming conventions	53
1.5.4 Procedural interface conventions	53
1.5.5 Specification of transmitted data	56
1.5.6 State diagram conventions	58
1.6 General considerations	59
1.6.1 Interface between equipment	59
1.6.2 Interface between vehicles	59
1.6.3 Real-Time Protocols	59
1.6.4 Network Management	60
1.6.5 Configurations	60
1.6.6 Structure of a standard device	61
1.6.6.1 TCN device	61
1.6.6.2 MVB options	61
1.6.6.3 WTB options	61
1.6.6.4 RTP options	61
1.6.6.5 TNM options	61
1.7 Conformance Test	64
2 Real-Time Protocols	65
2.1 General	65
2.1.1 Contents of this clause	65
2.1.2 Structure of this clause	66
2.2 Variables – Services and Protocols	67
2.2.1 General	67
2.2.2 Link layer Interface for Process_Data	67
2.2.2.1 Purpose	67
2.2.2.2 Datasets	67
2.2.2.3 Link_Process_Data_Interface primitives	69
2.2.3 Application interface for Process_Variables	73
2.2.3.1 Purpose	73
2.2.3.2 Process_Variables	73
2.2.3.3 Application_Variables_Interface primitives	77

Clause		Page
2.3	Messages Services and Protocols.....	87
2.3.1	General	87
2.3.2	Reference station.....	87
2.3.2.1	Terminal station	87
2.3.2.2	Router station	88
2.3.2.3	Station identifier	88
2.3.2.4	Bus identifier.....	88
2.3.2.5	Link address	89
2.3.3	Message packets handling	89
2.3.3.1	Packet pools	89
2.3.3.2	Type 'MD_PACKET'	89
2.3.3.3	Procedure Type 'MD_GET_PACKET'	90
2.3.3.4	Procedure Type 'MD_PUT_PACKET'	91
2.3.4	Message Link layer	91
2.3.4.1	Purpose	91
2.3.4.2	Link layer structure.....	92
2.3.4.3	Link layer characteristics	93
2.3.4.4	Link layer of the vehicle bus	94
2.3.4.5	Link layer of the train bus	95
2.3.4.6	Link layer interface for Message Data (LMI).....	96
2.3.5	Message Network Layer.....	103
2.3.5.1	Purpose.....	103
2.3.5.2	Directories	104
2.3.5.3	Network layer constants and variables.....	106
2.3.5.4	Network_Address.....	106
2.3.5.5	Network Header encoding	109
2.3.5.6	Routing in the network layer	113
2.3.5.7	Network layer interface.....	117
2.3.6	Message transport layer	118
2.3.6.1	Purpose	118
2.3.6.2	Message Transport Protocol.....	118
2.3.6.3	Packet exchange.....	119
2.3.6.4	Connection reference	121
2.3.6.5	Transport layer packets.....	121
2.3.6.6	State transition diagrams.....	129
2.3.6.7	Transport Message Interface	143
2.3.7	Multicast Transport Protocol (option)	150
2.3.7.1	Multicast packet exchange	150
2.3.7.2	Connection Identification	155
2.3.7.3	Connection reference	155
2.3.7.4	Multicast Message Transport Control encoding.....	155
2.3.7.5	Multicast Packets encoding	155
2.3.7.6	Multicast transport protocol definition	156

Clause		Page
2.3.8	Message session layer	167
2.3.8.1	Purpose	167
2.3.8.2	Conversation identifier	168
2.3.8.3	Shortcut	168
2.3.8.4	Topo_Counter check	168
2.3.8.5	Session header encoding	168
2.3.8.6	Buffer management.....	169
2.3.8.7	Session layer interface	169
2.3.9	Message Presentation Layer	169
2.3.10	Message Application layer.....	169
2.3.10.1	Purpose	169
2.3.10.2	Application_Messages_Interface	170
2.3.10.3	Caller side.....	174
2.3.10.4	Replier site.....	177
2.3.10.5	Initialisation.....	178
2.3.10.6	Station directory interface.....	179
2.3.10.7	Function directory interface	181
2.3.10.8	Group directory interface	182
2.3.10.9	Caller application interface.....	182
2.3.10.10	Replier application interface	186
2.3.10.11	Procedure 'am_buffer_free'	191
2.3.10.12	Multicast Application Interface	191
2.4	Presentation and encoding of transmitted and stored data.....	191
2.4.1	Purpose	191
2.4.2	Data ordering.....	191
2.4.2.1	Transmission format	191
2.4.2.2	Traffic_Store format	191
2.4.2.3	Application data format.....	191
2.4.2.4	General rules	192
2.4.2.5	Relationship with ASN.1	192
2.4.3	Notation for the primitive types	193
2.4.3.1	Notation for the boolean type.....	193
2.4.3.2	Notation for the antivalent type	194
2.4.3.3	Notation for the unsigned integer types.....	194
2.4.3.4	Notation for the integer type	195
2.4.3.5	Notation for the enumerated type.....	196
2.4.3.6	Notation for the binary coded decimal type	197
2.4.3.7	Notation for the unipolar types.....	197
2.4.3.8	Notation for the bipolar types.....	198
2.4.3.9	Notation for the real type	199
2.4.3.10	Notation for the character type	199
2.4.3.11	Notation for the unicode character type	199
2.4.3.12	Notation for the uncommitted types	200

Clause	Page
2.4.4 Structured types.....	200
2.4.4.1 General.....	200
2.4.4.2 Notation for the record types	201
2.4.4.3 Notation for the bitset types.....	201
2.4.4.4 Notation for the array type	203
2.4.4.5 Notation for the choice types	205
2.4.4.6 Notation for the set types	207
2.4.5 Alignment	209
2.4.6 Notation for special types	210
2.4.6.1 Notation for the string type	210
2.4.6.2 Notation for the TIMEDATE48 type.....	210
2.4.6.3 Notation for the TIME64 type.....	211
2.4.6.4 Notation for the ASN.1 boolean8 type	211
2.4.6.5 Notation for Little-Endian types.....	212
3 Multifunction Vehicle Bus	213
3.1 General	213
3.1.1 Contents of this clause.....	213
3.1.2 Structure of this clause	213
3.2 Physical Layer	215
3.2.1 Topology.....	215
3.2.1.1 Segments	215
3.2.1.2 Couplers.....	215
3.2.1.3 Double-line segments.....	216
3.2.2 Device classes.....	217
3.2.2.1 Capabilities	217
3.2.2.2 Class 0 devices.....	217
3.2.2.3 Class 1 devices.....	217
3.2.2.4 Class 2 devices.....	218
3.2.2.5 Class 3 devices.....	218
3.2.2.6 Class 4 devices.....	218
3.2.2.7 Class 5 devices.....	218
3.2.2.8 Device Attachment	218
3.2.3 Specifications common to all media.....	218
3.2.3.1 Signalling speed.....	218
3.2.3.2 Maximum propagation delay.....	218
3.2.3.3 Transceiver interface.....	219
3.2.3.4 Redundant medium (option)	220
3.2.4 Electrical Short Distance medium (choice).....	220
3.2.4.1 ESD topology	220
3.2.4.2 ESD configuration rules.....	221
3.2.4.3 ESD section specifications	223
3.2.4.4 ESD shielding	223
3.2.4.5 ESD medium-dependent interface	224

Clause		Page
	3.2.4.6 ESD Line_Unit specifications.....	227
	3.2.4.7 ESD signal wave form	227
	3.2.4.8 ESD transmitter.....	228
	3.2.4.9 ESD receiver.....	229
3.2.5	Electrical Middle Distance medium (choice).....	230
	3.2.5.1 EMD topology.....	230
	3.2.5.2 EMD configuration rules	230
	3.2.5.3 EMD terminator	231
	3.2.5.4 Cable section	231
	3.2.5.5 EMD shielding.....	232
	3.2.5.6 EMD medium-dependent interface.....	233
	3.2.5.7 EMD Line_Unit specifications	237
	3.2.5.8 EMD signal waveform.....	239
	3.2.5.9 EMD transmitter specifications	239
	3.2.5.10 EMD receiver specifications	242
3.2.6	Optical Glass Fibre medium (choice).....	244
	3.2.6.1 OGF topology.....	244
	3.2.6.2 OGF optical cable and fibre.....	245
	3.2.6.3 OGF medium-dependent interface.....	240
	3.2.6.4 OGF test signal (guideline).....	246
	3.2.6.5 OGF transmitter specifications	247
	3.2.6.6 OGF receiver specifications	248
	3.2.6.7 OGF active star coupler	248
	3.2.6.8 OGF double-line layout (option)	249
3.3	Medium-dependent signalling.....	250
3.3.1	Frame encoding and decoding	250
	3.3.1.1 Conventions	250
	3.3.1.2 Bit encoding	250
	3.3.1.3 Non-data symbols	251
	3.3.1.4 Start Bit	251
	3.3.1.5 Start Delimiter	251
	3.3.1.6 End Delimiter	252
	3.3.1.7 Valid frame (definition)	253
	3.3.1.8 Detection of line idle.....	253
	3.3.1.9 Detection of collision.....	253
	3.3.1.10 Receiver behaviour in case of error	253
	3.3.1.11 Jabber halt.....	254
3.3.2	Line Redundancy (option).....	254
	3.3.2.1 Principle.....	254
	3.3.2.2 Redundant transmission.....	254
	3.3.2.3 Redundant reception	255
	3.3.2.4 Switchover	255
	3.3.2.5 Redundancy status report.....	255

Clause		Page
3.3.3	Repeater	256
3.3.3.1	Repeater between single-line segments	256
3.3.3.2	Repeater for redundant medium (option)	257
3.4	Frames and Telegrams	258
3.4.1	Frame format	258
3.4.1.1	Master Frame format	258
3.4.1.2	Slave Frame format	258
3.4.1.3	Check Sequence	259
3.4.2	Telegram timing	259
3.4.2.1	Conventions	259
3.4.2.2	Reply delay (definition)	260
3.4.2.3	Frame spacing at the source	262
3.4.2.4	Frame spacing at the destination	262
3.4.2.5	Frame spacing at the master	262
3.5	Link Layer Control	264
3.5.1	Addressing	264
3.5.1.1	Device Address	264
3.5.1.2	Logical Address	264
3.5.1.3	Group Address	265
3.5.2	Master Frames contents	265
3.5.2.1	Master Frame format	265
3.5.2.2	F_code encoding	265
3.5.3	Slave Frame contents	267
3.5.3.1	Slave Frame format	267
3.5.3.2	Size error	267
3.5.4	Telegram types	267
3.5.4.1	Process Data telegrams	267
3.5.4.2	Message Data	268
3.5.4.3	Supervisory Data telegrams	269
3.6	Medium allocation	270
3.6.1	Organisation	270
3.6.1.1	Turn	270
3.6.1.2	Basic Period	271
3.6.1.3	Padding	271
3.6.2	Periodic Polling	271
3.6.2.1	Periodic List	271
3.6.2.2	Individual Period	271
3.6.2.3	Periodic Phase construction	273
3.6.3	Event Polling	274
3.6.3.1	Group Address	274
3.6.3.2	Event_Round	275
3.6.3.3	Recommended event search algorithm	277
3.6.3.4	Supervisory Data frames for Event_Arbitration	278

Clause		Page
3.6.4	Devices_Scan	280
3.6.4.1	Device_Status	280
3.6.4.2	Device_Status protocol	283
3.6.4.3	Devices_Scan protocol	283
3.7	Mastership transfer	283
3.7.1	Mastership transfer operation	283
3.7.1.1	Bus administrator configuration	283
3.7.2	Mastership transfer specifications	285
3.7.2.1	States	287
3.7.2.2	Time-outs for mastership transfer	287
3.7.2.3	Supervisory Data frames for mastership transfer	287
3.8	Link Layer Interface	288
3.8.1	Link Layer layering	288
3.8.2	Link Process Data interface	289
3.8.3	Link Message Data interface	289
3.8.3.1	General	289
3.8.3.2	Priority	290
3.8.3.3	Packet size	290
3.8.3.4	Protocol_Type	290
3.8.3.5	Message Transport Protocol	290
3.8.4	Link Supervision Interface	290
3.8.4.1	General	290
3.8.4.2	Link Supervision Interface procedures	291
3.8.4.3	MVB_Status	291
3.8.4.4	MVB_Control	292
3.8.4.5	MVB_Devices	293
3.8.4.6	MVB_Administrator	294
3.8.4.7	MVB_Report	297
4	Wire Train Bus	298
4.1	General	298
4.1.1	Contents of this clause	298
4.1.2	Structure of this clause	301
4.2	Physical layer	301
4.2.1	Topology	301
4.2.1.1	Bus sections	301
4.2.1.2	Couplers	301
4.2.1.3	Nodes	301
4.2.1.4	Vehicle orientation	302
4.2.1.5	Vehicle specification (informal)	302
4.2.2	Medium specifications	303
4.2.2.1	Topology	303
4.2.2.2	Duplicated medium (option)	304

Clause		Page
	4.2.2.3 Bus Configuration rules	304
	4.2.2.4 Cable specification	305
	4.2.2.5 Shielding concept	307
	4.2.2.6 Terminator	308
4.2.3	Medium attachment	308
	4.2.3.1 Node connection points identification	309
	4.2.3.2 Direct node attachment	309
	4.2.3.3 Indirect node attachment	309
	4.2.3.4 Connector (optional)	310
4.2.4	Node specifications	311
	4.2.4.1 Node elements	311
	4.2.4.2 Node and switch settings	313
	4.2.4.3 Duplicated Line Units (option)	313
4.2.5	Line Unit specifications	314
	4.2.5.1 Galvanic separation	314
	4.2.5.2 Insertion losses of a Line Unit	314
	4.2.5.3 Switches specifications	315
	4.2.5.4 Shield connection to a Line Unit	315
	4.2.5.5 Fritting (option)	316
4.2.6	Transceiver specifications	318
	4.2.6.1 Conventions	318
	4.2.6.2 Transmitter	318
	4.2.6.3 Receiver specifications	321
4.3	Medium-dependent signalling	323
4.3.1	Frame encoding and decoding	323
	4.3.1.1 Conventions	323
	4.3.1.2 Bit encoding	324
	4.3.1.3 Preamble encoding	324
	4.3.1.4 End Delimiter	325
	4.3.1.5 Signal quality supervision	325
4.3.2	Duplicated line handling (option)	327
	4.3.2.1 Principle	327
	4.3.2.2 Skew	327
	4.3.2.3 Redundant transmission	328
	4.3.2.4 Redundant reception	328
	4.3.2.5 Switchover	329
	4.3.2.6 MAU report	329
4.3.3	Line Unit interface	329
4.4	Frames and telegrams	330
4.4.1	Frame_Data format	331
4.4.2	Telegram timing	332
	4.4.2.1 Conventions	332
	4.4.2.2 Computation of the Reply Delay	332
	4.4.2.3 Collision	334
	4.4.2.4 Transmitted frame spacing	334

Clause		Page
4.5	Link Layer Control	335
4.5.1	Addressing.....	335
4.5.2	Frame structure	335
4.5.2.1	Elements of the HDLC Frame.....	335
4.5.2.2	Link Control Field	336
4.5.2.3	Handling of 'Attention', 'Change', 'Inhibit', and 'Remove_Inhibit'	340
4.5.2.4	Size, FCS and protocol errors.....	340
4.5.3	Telegram formats.....	340
4.5.3.1	Link Data field.....	340
4.5.3.2	Process Data	340
4.5.3.3	Message Data.....	342
4.5.3.4	Supervisory Data.....	343
4.5.3.5	Detection telegram.....	344
4.5.3.6	Presence telegram	346
4.5.3.7	Status telegram.....	347
4.5.3.8	Set to Intermediate telegram.....	349
4.5.3.9	Naming telegram	350
4.5.3.10	Unname telegram.....	352
4.5.3.11	Set to End telegram	353
4.5.3.12	Topography telegram.....	354
4.6	Medium allocation	357
4.6.1	Organisation	357
4.6.1.1	Basic Period.....	357
4.6.2	Periodic Phase	357
4.6.2.1	Individual Period	357
4.6.2.2	Periodic List	358
4.6.2.3	Rolling of the End Nodes	358
4.6.2.4	Error conditions and treatment	358
4.6.3	Sporadic phase	359
4.6.3.1	Event announcement.....	359
4.6.3.2	Message List.....	359
4.6.3.3	Supervisory List	359
4.6.3.4	Background scanning (option)	359
4.7	Inauguration	359
4.7.1	General.....	359
4.7.1.1	Address Allocation	359
4.7.1.2	Node ranking	360
4.7.2	Descriptors	361
4.7.2.1	Node Descriptor	361
4.7.2.2	Node Report	362
4.7.2.3	User Report	362
4.7.2.4	Composition strength	363

Clause		Page
	4.7.2.5 Master_Report	364
	4.7.2.6 Topo Counter	364
	4.7.2.7 Master Topo.....	364
	4.7.2.8 Inauguration_Counter.....	365
4.7.3	Detection of other compositions (informal).....	365
	4.7.3.1 Detection protocol	365
	4.7.3.2 Collision avoidance rules	366
4.7.4	State diagrams of the inauguration	368
	4.7.4.1 Node structure	368
	4.7.4.2 Inauguration process structure	368
	4.7.4.3 Specification language	370
	4.7.4.4 Node variables	370
	4.7.4.5 Auxiliary_Process.....	374
	4.7.4.6 Major states of the Main Process.....	379
	4.7.4.7 Macro 'START_NODE'	382
	4.7.4.8 Master States.....	383
	4.7.4.9 Slave States.....	400
	4.7.4.10 Time-outs	410
4.8	Link layer interface	411
	4.8.1 Link layer layering	411
	4.8.2 Link Process_Data_Interface.....	412
	4.8.2.1 General.....	412
	4.8.2.2 WTB specific.....	412
	4.8.3 Link Message_Data_Interface	413
	4.8.3.1 General.....	413
	4.8.3.2 Packet size	413
	4.8.3.3 Protocol_Type.....	413
	4.8.3.4 Message transport protocol	413
	4.8.4 Link management interface	413
	4.8.4.1 General.....	413
	4.8.4.2 Interface procedures	413
	4.8.4.3 Reporting	414
	4.8.4.4 Initialisation service.....	415
	4.8.4.5 Reset service	415
	4.8.4.6 Configuration service	416
	4.8.4.7 Set Slave service	418
	4.8.4.8 Set Weak service	418
	4.8.4.9 Set Strong service.....	419
	4.8.4.10 StartNaming service.....	419
	4.8.4.11 Remove service	419
	4.8.4.12 Inhibit service.....	419
	4.8.4.13 Allow service.....	419
	4.8.4.14 SetSleep service	420

Clause		Page
	4.8.4.15 CancelSleep service.....	420
	4.8.4.16 GetStatus service.....	420
	4.8.4.17 Get WTB nodes service	421
	4.8.4.18 Get Topography service	421
	4.8.4.19 Change Node Descriptor service	423
	4.8.4.20 Change User Report service.....	424
	4.8.4.21 Change Inauguration_Data service.....	424
	4.8.4.22 Get Statistics service	424
	4.8.4.23 Get Inauguration_Data service	425
5	Train Network Management	426
5.1	General	426
5.1.1	Contents of this clause.....	426
5.1.2	Structure of this clause	427
5.2	Manager, Agents and interfaces.....	427
5.2.1	Manager and Agent.....	427
5.2.2	Management messages protocol.....	427
5.2.3	Interfaces	428
	5.2.3.1 Object interface.....	428
	5.2.3.2 Manager interface (MGI)	430
	5.2.3.3 Agent interface (AGI).....	430
5.3	Managed objects	430
5.3.1	Object Attributes	430
5.3.2	Station objects	430
	5.3.2.1 Station_Status object	430
	5.3.2.2 Station_Control object.....	426
	5.3.2.3 Station Inventory object.....	432
	5.3.2.4 Station Reservation object.....	432
5.3.3	MVB link objects	438
	5.3.3.1 MVB_Status object.....	433
	5.3.3.2 MVB_Control object	433
	5.3.3.3 MVB_Configuration object	434
	5.3.3.4 MVB Devices object	434
5.3.4	WTB link objects	434
	5.3.4.1 WTB Status object	434
	5.3.4.2 WTB Topography object.....	435
5.3.5	Variable objects	435
	5.3.5.1 Port Configuration object.....	435
	5.3.5.2 Variables object	435
	5.3.5.3 Port_Attach	436
5.3.6	Messenger objects	437
	5.3.6.1 Messenger status.....	437
	5.3.6.2 Messenger control.....	437
	5.3.6.3 Directory objects	437

Clause		Page
5.3.7	Domain objects	438
5.3.7.1	Objects	438
5.3.7.2	Services.....	438
5.3.8	Task objects	439
5.3.8.1	Objects	439
5.3.8.2	Access.....	439
5.3.8.3	Services.....	439
5.3.9	Clock object.....	439
5.3.9.1	Objects	439
5.3.9.2	Services.....	439
5.3.10	Journal object	439
5.3.10.1	Object.....	439
5.3.10.2	Services.....	440
5.3.11	Equipment object	440
5.3.11.1	Object.....	440
5.3.11.2	Services.....	440
5.4	Services and management messages	440
5.4.1	Notation for all management messages.....	440
5.4.1.1	Message structure.....	440
5.4.1.2	Notation for the SIF_code.....	441
5.4.1.3	Notation for a call management message	442
5.4.1.4	Notation for a reply management message	443
5.4.1.5	Status and Error report.....	444
5.4.2	Station services	445
5.4.2.1	Read_Station_Status.....	445
5.4.2.2	Write_Station_Control	446
5.4.2.3	Read_Station_Inventory	447
5.4.2.4	Write_Station_Reservation	448
5.4.2.5	Read_Service_Descriptor	450
5.4.2.6	Read_Links_Descriptor	450
5.4.2.7	Write_Links_Descriptor	451
5.4.3	MVB link services.....	452
5.4.3.1	Read_MVB_Status	452
5.4.3.2	Write_MVB_Control.....	455
5.4.3.3	Read_MVB_Devices.....	456
5.4.3.4	Write_MVB_Administrator	457
5.4.4	WTB link services	465
5.4.4.1	Read_Wtb_Status	465
5.4.4.2	Write_Wtb_Control.....	468
5.4.4.3	Read_Wtb_Nodes	469
5.4.4.4	Read_Wtb_Topography.....	470
5.4.5	Variables services.....	472
5.4.5.1	Read_Ports_Configuration	472
5.4.5.2	Write_Ports_Configuration	473
5.4.5.3	Read_Variables.....	474

Clause		Page
	5.4.5.4 Write_Force_Variables.....	476
	5.4.5.5 Write_Unforce_Variables.....	478
	5.4.5.6 Write_Unforce_All.....	478
	5.4.5.7 Read_Variable_Bindings	479
	5.4.5.8 Write_Variable_Bindings	481
	5.4.5.9 Write_Attach_Port.....	482
5.4.6	Messages services.....	482
	5.4.6.1 Read_Messenger_Status	482
	5.4.6.2 Write_Messenger_Control.....	485
	5.4.6.3 Read_Function_Directory.....	487
	5.4.6.4 Write_Function_Directory.....	487
	5.4.6.5 Read_Station_Directory.....	488
	5.4.6.6 Write_Station_Directory	489
	5.4.6.7 Read_Group_Directory.....	490
	5.4.6.8 Write_Group_Directory.....	491
	5.4.6.9 Read_Node_Directory.....	492
	5.4.6.10 Write_Node_Directory.....	482
5.4.7	Domain services	493
	5.4.7.1 Read_Memory.....	493
	5.4.7.2 Write_Memory.....	495
	5.4.7.3 Download_Setup.....	496
	5.4.7.4 Download_Segment.....	498
5.4.8	Task services.....	499
	5.4.8.1 Read_Tasks_Status	499
	5.4.8.2 Write_Tasks_Control.....	500
5.4.9	Clock services	501
	5.4.9.1 Read_Clock	501
	5.4.9.2 Write_Clock	502
5.4.10	Journal Service	502
	5.4.10.1 Read_Journal.....	502
5.4.11	Equipment Service	504
	5.4.11.1 Read_Equipment.....	504
5.5	Interface Procedures	505
	5.5.1 Manager interface (MGI)	505
	5.5.2 Agent interface.....	506
	5.5.2.1 Description.....	506
	5.5.2.2 Agent control procedures	506
	5.5.2.3 User service subscription	507
	5.5.2.4 Restart procedure subscription.....	508

Tables

1	Template for the specification of an interface procedure	55
2	Example of message structure.....	56
3	Example of textual message form (corresponding to table 2).....	57
4	State transition table.....	58
5	Interoperability Testing	64
6	LPI primitives	70
7	Var_Size and Var_Type encoding in a PV_Name	76
8	LMI primitives.....	98
9	Routing situations.....	114
10	Routing of packets coming from the transport layer.....	115
11	Routing of packets coming from a vehicle bus.....	117
12	Routing of packets coming from the train bus.....	118
13	Message Transport Control encoding.....	123
14	Connect_Request.....	126
15	Connect_Confirm.....	126
16	Disconnect_Request.....	127
17	Disconnect_Confirm	127
18	DataPacket	127
19	Ack_Packet.....	128
20	Nak_Packet.....	128
21	Broadcast_Connect (BC1, BC2, BC3).....	128
22	Broadcast_Data.....	129
23	Broadcast_Repeat.....	129
24	Broadcast_Stop (BSC, BSO).....	130
25	MTP states.....	130
26	MTP incoming events	132
27	MTP outgoing events.....	132
28	MTP control parameters	133
29	MTP auxiliary variables.....	134
30	MTP time-outs (worst case)	136
31	Implicit actions	136
32	Compound actions.....	137
33	Producer states and transitions.....	138
34	Consumer states and transitions.....	141
35	TMP primitives	145
36	States of the MCP machine	157
37	Incoming Events.....	158
38	Outgoing Events.....	158
39	Control fields in packets	159
40	Auxiliary variables	160
41	MCP constants	161
42	MCP time-outs.....	161
43	MCP Compound actions	162
44	Filtering or BR packets	163

Tables (continued)

45	MCP Producer state event table.....	164
46	MCP Consumer state event table.....	166
47	AMI primitives.....	171
48	Address constants.....	173
49	System Address and User Address.....	176
50	MVB devices capabilities	217
51	Pin assignment for the ESD connector.....	226
52	Pin assignment for the EMD connector	236
53	Master Frame types and F_code.....	266
54	LS_RESULT encoding	291
55	MVB_Status object	292
56	MVB_Control object.....	293
57	MVB_Devices object.....	294
58	MVB_Administrator object	295
59	LS_V_REPORT encoding	297
60	WTB connector pin assignment	311
61	Signals of the Line Unit Interface	330
62	Link Control encoding.....	337
63	NodeControl data structure.....	371
64	MyStatus data structure	372
65	Shared Variables of a node.....	373
66	Variables of Main Process	374
67	Lists of main Process	374
68	'START_node'	380
69	'MASTER STATES'.....	380
70	'SLAVE STATES'.....	386
71	Time constant values.....	410
72	Example of mvb_administratof_list.....	463

Figures

1	Layering of the TCN	30
2	State transition example	58
3	Interfaces between equipment	59
4	Interfaces between vehicles.....	60
5	Train Bus and Vehicle Bus.....	61
6	TCN configurations.....	62
7	TCN device configuration options	65
8	Structure of the Train Communication Network	66
9	Real_Time Protocols layering	66
10	LPI primitives exchange.....	70
11	Check_Variable	75
12	Individual access.....	78
13	Set access	78
14	Cluster access.....	85

Tables (continued)

15	Terminal station.....	87
16	Router station between WTB and MVB	88
17	Packet format.....	90
18	Link layer data transmission	92
19	Link_Message_Data_Interface (LMI).....	93
20	Example of MVB Message_Data frame	95
21	Example of WTB Message_Data frame.....	96
22	LMI primitives	97
23	Network layer on a Node.....	104
24	Encoding of the Network_Address	107
25	Building of the addresses in an outbound packet.....	109
26	Network address encoding on the vehicle bus (example MVB)	111
27	Network address encoding on the train bus (example with WTB).....	113
28	Transport packet exchange	119
29	Packet formats (transport layer body)	122
30	State Transition diagram of the MTP.....	131
31	Time-out SENT_TMO	135
32	Time-out ALIVE_TMO.....	135
33	Transport interface	144
34	Multicast message with no retransmission	151
35	Short multicast message with no BD packets and no loss	152
36	Exchange with lost packets.....	153
37	Packet formats	156
38	Protocol machine states	157
39	Session layer transfer.....	168
40	Session_Header in Call_Message (of type Am_Result)	169
41	Application_Messages_Interface.....	170
42	Encoding of AM_ADDRESS.....	174
43	Reference device and structure of the document.....	214
44	MVB configuration	215
45	Transceiver interface.....	220
46	Example of ESD segment.....	221
47	Example of terminator	222
48	ESD Backplane section (double-line)	224
49	ESD connector arrangement.....	226
50	ESD terminator connector arrangement	227
51	Example of start of frame (ESD)	223
52	End of ESD frame (bothcases).....	229
53	EMD medium.....	230
54	Shielding (single-line segment)	232
55	Single-line device attachment	234
56	Double-line device attachment to EMD.....	235
57	EMD connectors arrangement.....	236
58	EMD terminator strapping	237
59	Measurement of an EMD device	238

Figures (continued)

60	Attenuation measurement.....	238
61	Example of start of an EMD frame	239
62	EMD Transmitter test circuits.....	240
63	Examples of pulse waveform at EMD transmitter.....	241
64	Example of end of EMD frame	242
65	EMD Receiver test signal.....	243
66	Optical link	244
67	Optical connector (dimensions in mm)	246
68	Example of start of OGF frame	247
69	Edge jitter.....	248
70	Example of active star coupler	249
71	Example of a duplicated star coupler	250
72	"0" and "1" data encoding	251
73	Non_Data symbols encoding.....	251
74	Master Start Delimiter.....	252
75	Slave Start Delimiter.....	252
76	Example of End Delimiter for EMD medium.....	252
77	Example of a valid frame (OGF medium).....	253
78	Signal skew.....	254
79	Example of repeater for single-line attachment.....	257
80	Example of repeater connecting a double-line to a single line segment.....	257
81	Master Frame Format.....	258
82	Slave Frames.....	258
83	Telegram timing.....	260
84	Example of Reply Delay.....	261
85	Frame spacing at the source side	262
86	Frame spacing at the destination(s)	262
87	Frame spacing at the master side	263
88	Master Frame contents.....	266
89	Word ordering in a Slave Frame	267
90	Process Data telegram	267
91	Message Data telegram.....	269
92	Supervisory Data telegram.....	270
93	Basic Periods	271
94	Example of constructin of the Macro_Cycle.....	274
95	General_Event_Request frame format	278
96	Group_Event_Request frame (M = 6, C = ABCDEF).....	279
97	Single_Event_Request frame.....	279
98	Event_Identifier_Response frame	280
99	Device_Status_Request.....	280
100	Device_Status_Response.....	281
101	Device_Status of Class 1 device	281
102	Device_Status of Class 2/3/4/5 device	282
103	Device_Status of a device with Bus Administrator capability.....	282

Figures (continued)

104	Device_Status of a device with Gateway capability	282
105	Mastership Transfer States	286
106	Device_Status_Request (sent by current master)	287
107	Device_Status_Response (sent by proposed master)	287
108	Mastership_Transfer_Request (sent by current master)	288
109	Mastership_Transfer_Response (sent by proposed next master)	288
110	Link Layer Layering	289
111	Wire Train Bus	298
112	Reference model of the WTB	300
113	Train Composition (two Intermediate Nodes shown)	301
114	Vehicle measurement	303
115	Connected nodes in regular operation	303
116	Double-line attachment	304
117	Grounded shield concept	307
118	Floating shield concept	308
119	Terminator	308
120	Direct node attachment (optional double-line)	309
121	Indirect attachment	310
122	WTB connector, front view	311
123	Example of MAU Structure	312
124	Node with redundant Line Units	314
125	Attenuation measurement	315
126	Shield grounding in the Line Unit	316
127	Fritting source and load	317
128	Transmitter test circuits	319
129	Pulse wave form at transmitter	320
130	Signal and idling at transmitter	321
131	Receiver signal envelope	322
132	Receiver edge distortion	323
133	Idealised frame on the line (16 bit Preamble shown)	324
134	Bit encoding	324
135	Preamble	325
136	End Delimiter	325
137	Valid frame, RxS, CS and SQE signals	326
138	Garbled frame RxS, CS and SQE signals	327
139	Redundant Lines (as seen at a receiver)	327
140	Line_Disturbance signals	328
141	HDLC Frame structure	331
142	Telegram timing	332
143	Example of Interframe spacing	333
144	Frame spacing measured at the master side	334
145	Frame spacing at the slave	334
146	HDLC Data format	335
147	Format of HDLC Data	336

Figures (continued)

148	Process Data telegram	340
149	Format of Process Data Request	341
150	Format of Process Data Response	342
151	Message Data telegram	342
152	Format of Message Data Request	342
153	Format of Message Data Response	343
154	Supervisory telegram	343
155	Detection telegram	344
156	Format of Detect Request	345
157	Format of Detect Response	345
158	Presence telegram	346
159	Format of Presence Request	346
160	Format of Presence Response	347
161	Status telegram	347
162	Format of Status Request	348
163	Format of Status Response	349
164	Set-to-Intermediate telegram	349
165	Format of SetInt Request	350
166	Format of SetInt Response	350
167	Naming telegram	351
168	Format of Naming Request	351
169	Format of Naming Response	352
170	Unnaming telegram	352
171	Format of Unname Request	352
172	Set to End telegram	353
173	Format of SetEnd Request	353
174	Format of SetEnd Response	354
175	Topography telegram	354
176	Format of Topography Request	355
177	Format of Topography Response	356
178	Structure of the Basic Period	357
179	Node position numbering	360
180	Format of Node Descriptor	361
181	Format of Node Report	362
182	Format of User Report	363
183	Format of Composition Strength	363
184	Master_Report	364
185	Format of Topo Counter	364
186	Format of Master Topo	365
187	Timing Diagram of detection protocol	367
188	Major node states and application settings	368
189	Node processes (End Setting)	369
190	AUXILIARY_PROCESS states	377
191	NAMING_RESPONSE macro	378

Figures (concluded)

192	States of MAIN PROCESS.....	379
193	Macro 'START_NODE'.....	382
194	Procedure REQUEST_RESPONSE.....	384
195	Procedures 'SET_TO_INIT' and 'SET_TO_END'	385
196	Macro 'INIT_MASTER'.....	386
197	Macro 'NAMING_MASTER'.....	387
198	Macro 'ASK_END'.....	388
199	Procedure NAME_ONE.....	391
200	Macro 'TEACHING_MASTER'.....	393
201	Macro 'UNNAMING_MASTER'	394
202	Macro 'REGULAR_MASTER'	396
203	Macro 'CHECK_DESC'	397
204	Macro 'PERIODIC_POLL'	399
205	Macro 'MESSAGE_POLL'	400
206	States 'UNNAMED_SLAVE'	402
207	States 'NAMED_SLAVE'	404
208	Macro 'LEARNING_SLAVE'	406
209	Macro 'REGULAR_SLAVE'	409
210	Link layer layering	411
211	Management messages.....	428
212	Agent Interfaces on a (gateway) Station.....	429
213	Station_Status.....	431
	Annex A (informative)	509
A.1	Introduction	509
A.1.1	Summary	509
A.1.2	Users Requirements.....	511
A.1.2.1	Data classes	511
A.1.2.2	Timing.....	511
A.1.2.3	Topology.....	511
A.1.2.4	Train_Bus	512
A.1.2.5	Vehicle_Bus.....	512
A.1.2.6	Availability.....	512
A.1.2.7	Environmental Condition.....	513
A.1.2.8	Classes of Busses.....	513
A.2	Tutorial on the Real-Time Protocols.....	514
A.2.1	Train Communication Network services.....	514
A.2.1.1	Periodic traffic and sporadic traffic	514
A.2.1.2	Transmitted Data	515
A.2.1.3	Variables and Messages	516
A.2.2	Tutorial on the Variables Services	519
A.2.2.1	General.....	519
A.2.2.2	Process_Variables	519
A.2.2.3	Process_Data protocol	523

Clause	Page
A.2.2.4 Datasets	525
A.2.2.5 PV_Name.....	527
A.2.2.6 Networking of Process_Variables	529
A.2.2.7 Application_Variables_Interface.....	532
A.2.3 Tutorial on the Messages Services	536
A.2.3.1 Reference architecture	536
A.2.3.2 Actual architecture	540
A.2.3.3 Messages Link Layer.....	541
A.2.3.4 Messages Network Layer	543
A.2.3.5 Messages Transport Layer	560
A.2.3.6 Messages Session_Layer.....	567
A.2.3.7 Messages Presentation_Layer.....	570
A.2.3.8 Messages Application_Layer_Interface (AMI).....	571
A.2.3.9 Software stack for Messages.....	571
A.2.4 Variables presentation.....	573
A.2.4.1 Variable types	573
A.2.4.2 Primitive and structured types	573
A.2.4.3 Error signalling.....	574
A.2.4.4 General rules	574
A.2.4.5 Type equivalence.....	576
A.2.5 Conformance to OSI.....	576
A.3 Tutorial on the Multifunction Vehicle Bus.....	578
A.3.1 Introduction	578
A.3.2 MVB Physical_Layer.....	579
A.3.2.1 Electrical Short Distance medium	580
A.3.2.2 Electrical Middle Distance medium	580
A.3.2.3 Optical medium	581
A.3.2.4 Bus segment connection	582
A.3.3 MVB devices	582
A.3.3.1 Bus_Controller	582
A.3.3.2 Device classes	583
A.3.3.3 Class 1 device.....	584
A.3.3.4 Class 2 and 3 devices	584
A.3.4 MVB signalling	585
A.3.5 MVB frames	586
A.3.6 MVB physical throughput.....	587
A.3.7 MVB Telegrams	588
A.3.7.1 Telegram Types	588
A.3.7.2 Process_Data Telegrams	589
A.3.7.3 Message_Data Telegrams.....	589
A.3.7.4 Supervisory_Data Telegrams.....	590
A.3.8 MVB Port	590
A.3.9 MVB medium allocation.....	591
A.3.9.1 Basic feasibility	592
A.3.9.2 Event Resolution	592

Clause	Page
A.3.10 MVB fault-tolerance	599
A.3.10.1 Fault-tolerance concept	599
A.3.10.2 Medium redundancy	599
A.3.10.3 Redundant Bus_Administrator	600
A.3.10.4 Token passing algorithm	601
A.4 Tutorial on the Wire_Train_Bus	602
A.4.1 WTB introduction	602
A.4.2 WT B topology	603
A.4.3 WT B medium	604
A.4.4 WT B Medium_Attachment_Unit	604
A.4.5 WTB Medium duplication	605
A.4.6 WTB signalling	606
A.4.7 WTB frames	607
A.4.7.1 Frame format	607
A.4.7.2 Frame integrity	607
A.4.8 WTB Telegrams	608
A.4.8.1 Process_Data Telegrams	608
A.4.8.2 Message_Data Telegrams	609
A.4.8.3 Supervisory_Data Telegrams	610
A.4.9 WT B medium access	610
A.4.10 WTB frames for regular operation (summary)	612
A.4.11 WTB Inauguration	612
A.4.11.1 Inauguration purpose	612
A.4.11.2 Node address assignment	612
A.4.11.3 Strong_Nodes and Weak_Nodes	613
A.4.11.4 Inauguration requirements	614
A.4.11.5 MAU Elements for inauguration	615
A.4.11.6 Named composition in regular operation	616
A.4.11.7 Detection of an unnamed node	616
A.4.11.8 Inauguration decision	617
A.4.11.9 Unnaming	617
A.4.11.10 Starting	618
A.4.11.11 Naming	618
A.4.11.12 Topography distribution	619
A.4.11.13 Master conflict	620
A.4.11.14 Weak_Master conflict	621
A.4.11.15 Strength comparison	622
A.4.11.16 Sample detection exchanges	622
A.4.11.17 Sample naming sequence	623
A.4.11.18 Master recovery	625
A.4.11.19 Bus shortening	626
A.4.11.20 Selection of the Master at power-up	627
A.4.11.21 Power-up, sleep mode and insertion	627
A.4.11.22 Quick reinsertion after temporary disruption (option)	628
A.4.11.23 Changes of Node_Descriptor	628

Clause	Page
A.4.12 WTB supervisory frames (summary)	630
A.4.13 WTB Master redundancy	631
A.5 Tutorial on the TCN Network_Management	633
A.5.1 Management role	633
A.5.2 Management phases	636
A.5.2.1 Testing	636
A.5.2.2 Commissioning	636
A.5.2.3 Operation	636
A.5.2.4 Maintenance	636
A.5.2.5 Example of a network configuration	636
A.5.2.6 Managed devices	637
A.5.3 Manager and Agent	638
A.5.4 Message protocol	638
A.5.5 Management Message format	639
A.5.6 Managed Objects	640
A.5.6.1 Overview	640
A.5.7 Related standards	641
A.6 Tutorial on the Guidelines for Conformance Test	642
A.6.1 Elements of Conformance Test	642
A.6.2 Limits of Conformance Testing	642
A.6.3 Types of test	642
A.6.4 Steps towards a conformance test definition	643
A.6.5 Reference documents	643
Annex B (normative)	644
B.1 General	644
B.1.1 Objectives	644
B.1.1.1 Definition of Conformance Test	645
B.1.1.2 General concept	646
B.1.1.3 Limits of conformance testing	646
B.1.2 Normative references	646
B.1.3 Definitions	647
B.1.3.1 Basic terms	647
B.1.3.2 Types of testing	648
B.1.3.3 Terminology of test suites	648
B.1.3.4 Terminology of results	649
B.1.3.5 Terminology of test methods	650
B.1.4 Symbols and abbreviations	652
B.2 Conformance test overview	653
B.2.1 Introduction	653
B.2.1.1 OSI model	653
B.2.1.2 Conformance test of an entity	654
B.2.1.3 Types of tests	656
B.2.2 Steps toward a conformance test definition	656

Clause	Page
B.2.2.1 Development of an abstract test suite (ATS)	656
B.2.2.2 Implementation of the test scripts	657
B.2.2.3 Execution of test sequences	657
B.2.2.4 Evaluation of the test	657
B.2.2.5 Certified laboratories	658
B.2.2.6 TCN Conformance Test	658
B.2.2.7 Conformance classes	659
B.2.2.8 TCN test functions	659
B.2.2.9 Maintenance of test procedures	660
B.3 Conformance test of an MVB device	660
B.3.1 Device identification (PICS Proforma)	660
B.3.1.1 Introduction	660
B.3.1.2 General identification of IUT	661
B.3.1.2 Physical interface characteristics	662
B.3.1.3 Link_Layer capabilities	663
B.3.1.4 Implemented communication objects	663
B.3.2 Physical interface	664
B.3.2.1 Test bed	664
B.3.2.2 Cable test	665
B.3.2.3 Connector test	666
B.3.2.4 Basic interconnection test	666
B.3.2.5 Frame transfer tests	670
B.3.2.6 Redundancy tests	670
B.3.3 Link_Layer test	670
B.3.3.1 Test bed	670
B.3.3.2 Tests list (IUT as slave)	671
B.3.3.3 Tests list (IUT as Bus_Administrator)	673
B.3.4 Real time protocols	673
B.3.4.1 Foreword	673
B.3.4.2 Transport layer	673
B.3.4.3 Session layer	674
B.3.4.4 Application interface	675
B.4 Conformance test of a WTB device	676
B.4.1 Device identification (PICS Proforma)	676
B.4.1.1 Introduction	676
B.4.1.2 General identification of IUT	676
B.4.1.3 Physical interface characteristics	676
B.4.1.4 Link_Layer capabilities	678
B.4.1.5 Implemented communication objects	678
B.4.2 Physical interface	679
B.4.2.1 Test bed	679
B.4.2.2 Cable test	681
B.4.2.3 Connector test	682
B.4.2.4 Signal tests	683
B.4.2.5 Switch tests	684

Clause	Page
B.4.2.6 Line fritting tests	684
B.4.2.7 Basic interconnection test.....	684
B.4.2.8 Frame transfer tests.....	684
B.4.2.9 Redundancy tests.....	685
B.4.3 Link_Layer test	685
B.4.3.1 Test bed.....	685
B.4.3.2 Slave test group	686
B.4.3.3 master test group	689
B.4.3.4 Redundancy tests inauguration on redundant lines	691
B.4.4 Real time protocols	693
B.4.4.1 Test bed.....	693
B.5 Conformance test of a gateway device.....	693
B.5.1 Device identification (PICS Proforma).....	693
B.5.1.1 Introduction.....	693
B.5.1.2 General identification of IUT	693
B.5.1.3 WTB interface characteristics.....	694
B.5.1.4 MVB interface characteristics.....	694
B.5.1.5 Implemented communication objects.....	694
B.5.2 Physical interfaces and Link_Layers.....	695
B.5.2.1 Test bed.....	695
B.5.2.2 Recommended tests.....	695
B.5.3 Real time protocol.....	695
B.5.3.1 Testing routing functionality of RTP in gateways	695
B.5.3.2 Test bed.....	695
B.5.3.3 Test of logical addressing.....	697
B.5.3.4 Test of physical addressing	697
B.5.3.5 Further router testing.....	697
B.6 Conformance test of a WTB.equipped vehicle.....	697
B.6.1 Vehicle identification	698
B.6.1.1 Introduction.....	698
B.6.1.2 General identification of IUT	698
B.6.2 Test bed	699
B.6.3 Physical interface	699
B.6.4 WTB Link_Layer capabilities	701
B.6.5 MVB interoperability test.....	701
B.6.6 Application profile	701
B.7 General testing requirements.....	702
B.7.1 Environment.....	702
B.7.1.1 Instrumentation	702
B.7.1.2 Test documentation.....	704

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**ELECTRIC RAILWAY EQUIPMENT –
TRAIN BUS –****Part 1: Train Communication Network****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 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.

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this International Standard may involve the use of a patent concerning ASICs and TCN software given in this standard.

The IEC takes no position concerning the evidence, validity and scope of this patent right.

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

JDP (a Joint Development Project executed by a group of designers from Siemens, Adtranz and FIREMA).

Lucent Technology : Ref: ASIC MBBD_3 and SDSP.

VLSI Technology GmbH: Ref. MVBC_01 Purchase terms and conditions.

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

International Standard IEC 61375-1 has been prepared by IEC Technical Committee 9: Electric railway equipment.

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

FDIS	Report on voting
9/511/FDIS	9/524/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.

Annex A is for information only.

Annex B is part of this standard.

According to subclause 5.8 of Part 2 of the ISO/IEC Directives, the following items are to be taken into consideration:

- In France the needs of data communication between equipment in vehicles and between vehicles are dealt with using products covered by other standards or technical specifications (many trains introduced in the last 10 years are equipped with data communication systems; care has been taken to make use as much as possible of different relevant industrial standards) incompatible with the provisions of this standard and making it inapplicable in that country.

For instance, in modern rolling stock recently commissioned in France, technical specifications for train communication network comply with IEC 61158-2 and EN 50170, which are the field bus international standards. On the other hand, ISO/IEC 8802-4 has been used in technical specifications for data communication networks for the latest generation of high speed trains.

- In China, for closed trains and multiple units, the technical specifications of a train communication network may be based on an agreement between user and manufacturer alternatively to this standard.

The committee has decided that this publication remains valid until 2004. At this date, in accordance with the committee's decision, the publication will be

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

INTRODUCTION

This International Standard defines interfaces so as to achieve plug-in compatibility:

- a) between equipment located in different vehicles, and
- b) between equipment located within the same vehicle.

This Standard defines these interfaces as connections to a data communication network, called the Train Communication Network (TCN).

The TCN has a hierarchical structure with two levels of busses, a Train Bus and a Vehicle Bus:

- a) for interconnecting vehicles in Open Trains (see definition) such as international UIC trains, this Standard specifies a Train Bus called the Wire Train Bus (WTB);
- b) for connecting standard on-board equipment, this standard specifies a Vehicle Bus called the Multifunction Vehicle Bus (MVB).

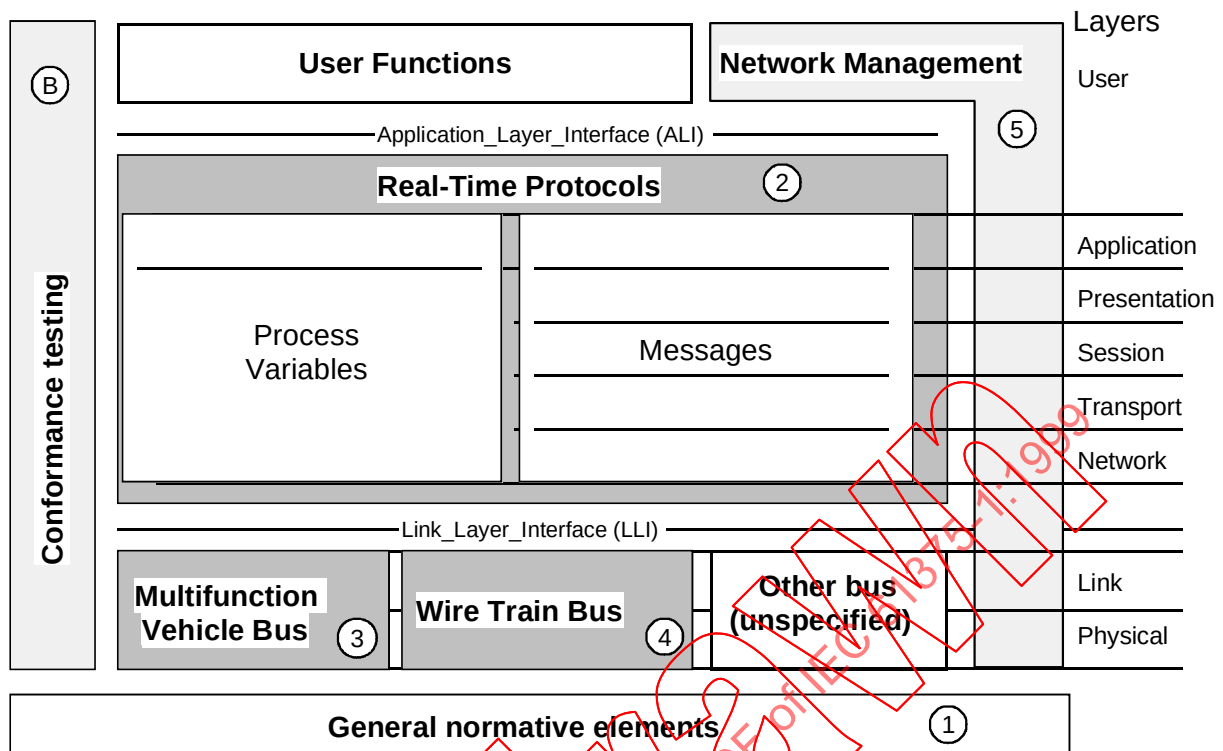
In the TCN architecture, all busses share the same Real-Time Protocols, which offer two communication services:

- a) Process Variables, a distributed, real-time database, periodically refreshed through broadcasting;
- b) messages, transmitted on demand either as:
 - unicast messages (point-to-point) or/and
 - multicast messages.

All busses in the TCN share a common Network Management, which allows debugging, commissioning and maintenance over the network.

Guidelines for conformance testing are included in this standard.

The TCN is structured similarly to the Open System Interconnection model defined in ISO/IEC 7498-1 (see Figure 1).



NOTE The circled numbers refer to the clauses and annexes of this standard.

Figure 1 – Layering of the TCN

This standard has been, for editorial reasons, divided into five clauses and two annexes:

Clause 1: General,

- definitions and informative overview;

Clause 2: Real-Time Protocols,

- Variables: Link Layer Interface and Application Layer Interface;
- Messages: Link Layer Interface, Protocols, Application Layer Interface;
- Data Representation;

Clause 3: Multifunction Vehicle Bus,

- Physical Layer, Link Layer and Link Layer Management;

Clause 4: Wire Train Bus,

- Physical Layer, Link Layer and Link Layer Management;

Clause 5: Train Network Management,

- configuration, supervision and control of the network;

Annex A: Tutorial on the Train Communication Network

Annex B: Guidelines for Conformance Test.

ELECTRIC RAILWAY EQUIPMENT – TRAIN BUS –

Part 1: Train Communication Network

1 General

1.1 Scope

This International Standard applies to data communication in Open Trains, i.e. it covers data communication between vehicles of the said open trains and data communication within the vehicles of the said open trains.

The applicability of this standard to the train communication bus (WTB) allows for interoperability of individual vehicles within Open Trains in international traffic. The data communication bus inside vehicles (MVB) is given as recommended solution to cope with the said TCN. In any case, proof of compatibility between WTB and a proposed vehicle bus will have to be brought by the supplier.

This standard may be additionally applicable to closed trains and multiple unit trains when so agreed between purchaser and supplier.

NOTE 1 For a definition of Open Trains, Multiple Unit Trains and Closed Trains, see 1.3.

NOTE 2 Road vehicles such as buses and trolley buses are not considered in this standard.

1.2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of IEC 61375. 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 61375 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 ISO and IEC maintain registers of currently valid International Standards.

IEC 60096-1: *Radio-frequency cables - Part 1: General requirements and measuring methods*

IEC 60245-1: *Rubber insulated cables – Rated voltages up to and including 450/750 V – Part 1: General requirements*

IEC 60304: *Standard colours for insulation for low-frequency cables and wires*

IEC 60332-1: *Tests on electric cables under fire conditions - Part 1: Test on a single vertical isolated wire or cable*

IEC 60571: *Electronic equipment used on rail vehicles*

IEC 60794-1-1: *Optical fibre cables - Part 1-1: Generic specification - General*

IEC 60807 (all parts): *Rectangular connectors for frequencies below 3 MHz*

IEC 60807-2, *Rectangular connectors for frequencies below 3 MHz - Part 2: Detail specification for a range of connectors, with assessed quality, with trapezoidal shaped metal shells and round contacts - Fixed solder contact types*

IEC 60870 (all parts), *Telecontrol equipment and systems*

IEC 60870-5-1, *Telecontrol equipment and systems - Part 5: Transmission protocols - Section 1: Transmission frame formats*

IEC 60874-6, *Connectors for optical fibres and cables - Part 6: Sectional specification for fibre optic connector - Type LSA*

IEC 60874-10, *Connectors for optical fibres and cables - Part 10: Sectional specification - Fibre optic connector type BFOC/2,5*

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

ISO/IEC 3309, *Information technology – Telecommunications and information exchange between systems - High-level data link control (HDLC) procedures - Frame structure*

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

ISO/IEC 8073, *Information technology - Open Systems Interconnection – Protocol for providing the connection-mode transport device*

ISO/IEC 8482, *Electrical characteristics of generators and receivers for use in balanced digital multipoint systems*

ISO/IEC 8802-2, *Local Area Networks - Logical Link Control*

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

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

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

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

ISO/IEC 9506-1 *Industrial automation systems – Manufacturing Message Specification – Part 1: Service definition*

ISO/IEC 9506-2 Industrial automation systems – Manufacturing Message Specification – Part 2: Protocol specification

ISO/IEC 9646 (all parts) Information technology – Open Systems Interconnection – Conformance testing methodology and framework

ISO/IEC 10646-1 Information Technology - Universal Multiple - octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Planes

ITU-T V24, *List of definitions for interchange circuits between data terminal equipment (DTE) and data-circuit terminating equipment (DCE)*, CCITT Recommendations Fascicle VIII.1 Rec. V24

ITU Z100 Annexe D, *SDL predefined data*, 1993

UIC 556 ORE B 108.3 Fiche No. 556, *"Transmission d'informations dans le train - bus de train"*, 12th edition

UIC 557, *"Diagnosis Techniques for Coaches"*, 1st Edition Jan 1992

UIC 558, *"Information Cable of the RIC-Coaches"*, 4th Edition Dec 1994

IEEE 754, *Standard for Binary Floating-Point Arithmetic*, IEEE, 1985

1.3 Terms and definitions

For the purpose of this International Standard, the following definitions apply.

NOTE Keywords in this Standard are written with the first letter of each word in upper case and, when they are composed of two or several words, these are joined by an underscore. This convention allows keywords to be tracked in the documents.

1.3.1 address

identifier of a communication partner, of which several types exist, depending on the layer.

1.3.2 Agent

application process in a Station which accesses the local managed objects on behalf of the Manager

1.3.3 Aperiodic Data

transmission of Process Data on a demand basis. This service is not used

1.3.4 Application Layer

upper layer in the OSI model, interfacing directly to the Application

1.3.5 Application Layer Interface

definition of the services offered by the Application Layer

1.3.6

Application Messages Adapter

code directly called by the application implementing the Messages services

1.3.7

Application Messages Interface

definition of the Messages services

1.3.8

Application Process

communicating entity, implemented for instance by a task

1.3.9

Application Processor

processor which runs a communicating Application Process

1.3.10

Application Supervision Interface

definition of the Supervision services available in particular to the Agent

1.3.11

Application Variables Adapter

code directly called by the application implementing the Variables services

1.3.12

Application Variables Interface

definition of the Variables services

1.3.13

arbiter

device, or common protocol followed by several devices, which selects one of several devices competing for mastership

1.3.14

Auxiliary Channel

channel used for detecting additional Nodes

1.3.15

Basic Period

bus activity is divided into periods. The shortest is the Basic Period, which consists of a Periodic Phase (for Periodic Data) and of a Sporadic Phase (for Message Data and Supervisory Data)

1.3.16

big-endian

ordering scheme for storing or transmitting data in which the most significant part of a multiple-octet data is stored at the lowest octet address, and transmitted first

1.3.17

bit-stuffing

method specified by ISO/IEC 3309 to prevent Frame Data from being misinterpreted as a Flag, consisting of inserting an additional "0" symbol after each string of five "1" symbols and removing this "0" at reception

1.3.18**bridge**

device which stores and forwards frames from one bus to another on the base of their Link Layer addresses

1.3.19**broadcast**

nearly simultaneous transmission of the same information to several destinations. Broadcast in the TCN is not considered reliable, i.e. some destinations may receive the information and others not

1.3.20**bus**

communication medium which broadcasts the same information to all attached participants at nearly the same time, allowing all devices to obtain the same sight of its state, at least for the purpose of arbitration

1.3.21**Bus Administrator**

device capable of becoming Master of the MVB

1.3.22**Bus Controller**

processor or integrated circuit in charge of the Link Layer of communication

1.3.23**Bus Switch**

switch or relay within a WTB Node which connects electrically the cable sections of the two directions

1.3.24**Caller**

Application Process which initialises a message exchange

1.3.25**Check Sequence**

method of error detection based on appending to the transmitted useful data a checksum or a cyclic redundancy check (CRC) calculated on the useful data

1.3.26**Check Variable**

Process Variable of type antivalent boolean protecting another Process Variable

1.3.27**Check Offset**

bit offset of a Check Variable within a Dataset

1.3.28**Closed Train**

train consisting of a set of vehicles, where the composition does not change during normal operation, for instance metro, suburban train, or high-speed train units

1.3.29**composition**

number and characteristics of the vehicles forming a train

**1.3.30
configuration**

definition of the topology of a bus, the devices connected to it, their capabilities and the traffic they produce; by extension, the operation of loading the devices with the configuration information before going to regular operation

**1.3.31
Connect Confirm**

response of the Consumer to the Connect Request of the Producer

**1.3.32
Connect Request**

first packet of a message sent from Producer to Consumer

**1.3.33
consistency**

Dataset consisting of several elements is consistent if all elements are read or written in one indivisible operation

**1.3.34
Consumer**

receiver of a message at the Transport Layer (see: Producer)

**1.3.35
continuity vehicle**

vehicle without an operational Train Bus Node, but carrying a section of the bus to connect passively the Train Bus of its adjacent vehicles.

**1.3.36
conversation**

data exchange at the Application Layer, consisting of a Call Message and a Reply Message (the latter is missing in the multicast protocol). A conversation starts with the first Connect Request frame and ceases when the last acknowledgement for the Reply Message has been received or is no longer expected

**1.3.37
datagram**

frame containing all information necessary to forward it to its final destination, without knowledge of previous frame's contents. Datagrams do not use a previous connection establishment and they are not acknowledged at the Link Layer

**1.3.38
Dataset**

all Process Variables transmitted in one Process Data frame

**1.3.39
delimiter**

sequence of signals which includes code violation symbols (neither "1" nor "0") which is used to delimit the start (Start Delimiter) and the end (End Delimiter) of a frame, as defined for instance in IEC 61158-2

**1.3.40
Destination Device**

receiver of a frame at the Link Layer (see: Source Device)

1.3.41**device**

unit connected to one or more busses

1.3.42**Device Address**

Device Address identifies a device within a bus; On the MVB, the Device Address has 12 bits; On the WTB, the Device Address has 8 bits, the least significant 6 bits being the Node Address; A device connected to several busses may have a different Device Address for each bus. Special devices such as repeaters only participate at the Physical Layer and have no Device Address

1.3.43**Device Status**

16-bit word expressing the capabilities and status of an MVB device connection

1.3.44**Direction 1**

one direction of a WTB Node

1.3.45**Direction 2**

other direction of a WTB Node

1.3.46**Electrical Middle Distance**

one of the media of the MVB

1.3.47**Electrical Short Distance**

one of the media of the MVB

1.3.48**End Delimiter**

sequence which ends a frame before the medium returns to idle

1.3.49**End Node**

Node which terminates the two bus segments connected to it but does not establish continuity between them

1.3.50**Event Round**

sequence of polls in which all events pending at the start are read

1.3.51**extension box**

wiring box where the trunk cable is interrupted and passively extended by an extension cable to connect a device

1.3.52**extension cable**

cable inserting a Node in a trunk cable, consisting of two separate twisted wire pairs per line, possibly of smaller cross-section than the trunk cable itself

1.3.53

field device

device attaching simple sensors and actuators to the bus, outside a rack

1.3.54

final

receiver of a packet (data or acknowledgement) at the Network Layer. When two devices communicate within the same bus, the final is located in the destination device (see: origin)

1.3.55

Flag

sequence of "1" and "0" symbols which serves to delimit the beginning or the end of a frame. Flags which would appear in the transmitted data are modified by bit-stuffing, as defined for instance in ISO/IEC 3309

1.3.56

frame

sequence of consecutive symbols sent in one time slot by a transmitter, between two slots where the line is idle.

1.3.57

Frame Check Sequence

16-bit FCS specified in ISO/IEC 3309

1.3.58

Frame Data

data transmitted between the Start Delimiter and the End Delimiter (on the MVB) or between the Preamble and the End Delimiter (on the WTB)

1.3.59

fritting

electrical cleaning of oxidised contacts by applying a breakdown voltage over the contact

1.3.60

Function

Application Process which exchanges messages with another Application Process

1.3.61

Function Directory

directory which maps a Function Identifier to a Station Identifier and vice-versa

1.3.62

Function Identifier

8-bit identifier of a Function

1.3.63

F code

in a Master Frame, indicates the request and the expected response Slave Frame size

1.3.64

gateway

connection between different busses at the Application Layer requiring application-dependent data analysis and protocol conversion

1.3.65**Group Address**

address of a multicast group to which a Node belongs

1.3.66**Group Directory**

directory which indicates to a Node to which multicast group it pertains

1.3.67**hamming distance**

minimum number of bits of a given correct bit sequence, which, if inverted, create a false bit sequence indistinguishable from a correct one

1.3.68**HDLC**

High-level Data Link Control, a set of standardised protocols, including ISO/IEC 3309 for data transmission

1.3.69**HDLC Data**

data transmitted in an HDLC frame

1.3.70**Inauguration**

operation executed in case of composition change, which gives all Nodes of the WTB their address relative to the Master, their orientation and the descriptor of all named Nodes on the same bus

1.3.71**Individual Period**

interval between two successive transmissions of the same Process Data from the same source. The Individual Period is a power-of-2 multiple of the Basic Period

1.3.72**instance**

a) one of several objects which share the same definition (object instance)

b) one of several (simultaneous or not) executions of the same program (process instance)

1.3.73**integrity**

property of a system to recognise and to reject wrong data in case of malfunction of its parts

1.3.74**Intermediate Node**

Node which establishes continuity between two bus sections connected to it, but does not terminate them

1.3.75**jumper cable**

cable connecting the trunk cables of two consecutive vehicles, possibly of a larger cross-section than the trunk cable, and which is plugged by hand in the case of the UIC-cable. There are generally two jumper cables between vehicles

1.3.76**Line**

non-redundant bus. A dual-thread bus consists of two lines

1.3.77

Line Unit

all circuits providing the electrical attachment to a line

1.3.78

Link Address

address supplied to the Link Layer to identify to which Bus and to which Device Address a packet is sent or received

1.3.79

Link Control

field in the HDLC frame which indicates the type of frame

1.3.80

Link Data

data transported by the Link Layer, but not relevant to it

1.3.81

Link Header

part of a Message Data frame relevant to the Link Layer

1.3.82

Link Layer

layer in the OSI model establishing point-to-point and broadcast connections between devices attached to the same bus

1.3.83

Link Layer Interface

interface between Link Layer and higher communication layers

1.3.84

Link Layer Management

interface controlling the Link Layer for management purposes

1.3.85

little-endian

ordering scheme for storing or transmitting data in which the least significant part of a multiple-octet data is stored at the lowest octet address, and transmitted first

1.3.86

local area network

part of a network characterised by a common medium access and address space

1.3.87

logical link control

protocols and associated frame formats which serve to control the Link Layer

1.3.88

Logical Address

address which is not bound to a specific device (e.g. the Process Data address)

1.3.89

Logical Port

ports of a device used for the Process Data traffic and addressed by the Logical Address

1.3.90**Macro Cycle**

number of Basic Periods corresponding to a Macro Period

1.3.91**Macro Period**

longest Individual Period, after which the periodic traffic returns to the same pattern, counted in milliseconds

1.3.92**Main Channel**

channel over which the main bus traffic is received

1.3.93**Management Message**

message exchanged between a Manager and an Agent for Network Management.

1.3.94**Manager**

Function in a Station which is dedicated to Network Management and which send management Call Messages through System Addresses

1.3.95**marshalling**

allocation of application addresses or names to the Process Variables of a dataset, that, on the WTB, depends on the Node Type and Version.

1.3.96**Master**

device which spontaneously sends information on a bus to a number of slave devices. It may give a Slave the right to transmit for one Slave Frame only within a limited time

1.3.97**Master Frame**

frame sent by a Master

1.3.98**Master Start Delimiter**

Start Delimiter of a Master Frame

1.3.99**medium access control**

sublayer of the Link Layer, which controls the access to the medium (arbitration, mastership transfer, polling)

1.3.100**medium dependent interface**

mechanical and electrical interface between the transmission medium and a Medium Attachment Unit

1.3.101**medium**

physical carrier of the signal: electrical wires, optical fibres, etc.

1.3.102

Medium Attachment Unit

device used as a coupler to the transmission medium

1.3.103

message

data item transmitted in one or several packets

1.3.104

Messages

transmission service of the TCN

1.3.105

Message Data

data transmitted sporadically by the Link Layer in relation to message transmission; the corresponding Link Layer service

1.3.106

messenger

communication stack caring for end-to-end message communication and interfacing to the application

1.3.107

multicast

transmission of the same message to a group of Repliers, identified by their Group Address. The word "multicast" is used even if the group includes all Repliers

1.3.108

Multifunction Vehicle Bus (MVB)

Vehicle Bus to be used for connecting programmable stations and simple sensors/actors.

1.3.109

Multiple Unit Train:

train consisting of a set of closed trains, where the composition of the set may change during normal operation

1.3.110

network

set of possibly different communication systems which interchange information in a commonly agreed way

1.3.111

Network Address

address which identifies a Function or a Station within the network. It can be either a User Address or a System Address

1.3.112

Network Header

part of a Message Data frame relevant to the Network Layer

1.3.113

Network Layer

layer in the OSI model responsible for routing between different busses

1.3.114**Network Management**

operations necessary to remotely configure, monitor, diagnose and maintain the network

1.3.115**Node**

device on the Wire Train Bus, which may act as gateway between Train Bus and Vehicle Bus

1.3.116**Node Address**

address of a Node on the Train Bus (6 bits). It is equal to the least significant 6 bits of the 8-bit Device Address on the WTB

1.3.117**Node Descriptor**

24-bit data structure which indicates for a Node its Node Period and its Node Key

1.3.118**Node Directory**

directory which maps the Node Address to the Device Address (one-to-one mapping in WTB)

1.3.119**Node Key**

16-bit identifier selected by the application to identify a Node's type and version. The Master distributes it to all other Nodes after each composition change and before exchanging data

1.3.120**Node Period**

on the WTB, desired Individual Period of a Node (identical to Individual Period except if overload occurs)

1.3.121**octet**

8-bit word stored in memory or transmitted as a unit *

1.3.122**Open Train**

train consisting of a set of vehicles where the configuration may change during normal operation, for instance international UIC trains

1.3.123**Optical Glass Fibre**

one of the media of the MVB

1.3.124**origin**

sender of a packet (data or acknowledgement) at the Network Layer. When two devices communicate within the same bus, the Origin is located on the source device (see: final)

* IEC prescribes 'octet' instead of 'byte'

1.3.125

packet

unit of a message (information, acknowledgement or control) transmitted in exactly one Message Data frame

1.3.126

period

time unit after which a periodic pattern repeats itself

1.3.127

Periodic Data

Process Data transmitted periodically, at an interval which is the Individual Period

1.3.128

Periodic List

list of Nodes, addresses or devices to be polled in each period of a Macro Cycle

1.3.129

Periodic Phase

phase during which the Master polls for Periodic Data, according to its Periodic List

1.3.130

Physical Address

Device Address on the MVB and the Node Address on the WTB which identify communicating devices on the same bus

1.3.131

Physical Port

Port used for the Message Data or the Supervisory Data traffic and addressed by the Device Address

1.3.132

pitch

distance between adjacent devices on the same electrical bus required to avoid clustering of bus loads

1.3.133

polling

sending of a Master Frame in order to receive a Slave Frame

1.3.134

Port

memory structure which contains data for transmission or reception, and in which a new value overwrites the former value (buffer, not queue). A Port provides means for simultaneous access by the bus and the application(s)

1.3.135

Port Index Table

look-up table which deduces the memory address of a port from the Logical Address of the Process Data

1.3.136**Preamble**

sequence of signals heading a frame for the purpose of synchronising the receiver, used on the WTB

1.3.137**Presentation Layer**

layer in the OSI model responsible for data representation and conversion

1.3.138**Process Data**

source-addressed data broadcast periodically by the link layer in relation with Process Variables transmission; the corresponding Link Layer service

1.3.139**Process Variable**

variable expressing the state of a process (e.g. speed, brake command)

1.3.140**Producer**

sender of a message at the Transport Layer (see: Consumer)

1.3.141**Publisher**

source of a Dataset for broadcasting (see: Subscriber)

1.3.142**PV Name**

identifier of a Process Variable

1.3.143**PV Set**

set of Process Variables belonging to the same Dataset

1.3.144**queue**

memory storing an ordered set of frames in a first-in, first-out fashion

1.3.145**rack**

equipment containing one or more devices, attached to the same segment

1.3.146**reassembly**

act of regenerating a long message from several packets generated by segmentation

1.3.147**receiver**

electronic device which may receive signals from the physical medium

1.3.148**Receive Queue**

queue for receiving Message Data in a device

1.3.149

regular operation

normal bus activity as opposed to Inauguration (WTB) or configuration (MVB)

1.3.150

repeater

connection at the Physical Layer between bus segments, providing an extension of the bus beyond the limits permitted by passive means. The connected segments operate at the same speed and with the same protocol. The delay introduced by a repeater is in the order of one bit duration

1.3.151

Replier

Application Process which has been requested by the Caller to receive a Call Message and to reply with a Reply Message

1.3.152

residual error rate

probability of integrity breach (unrecognised wrong bit) per transmitted bit

1.3.153

router

connection between two busses at the Network Layer, which forwards datagrams from one bus to another on the base of their Network Address

1.3.154

scan

polling of devices in a certain sequence for supervisory purposes

1.3.155

section

part of a segment, which is passively connected to another section without terminator in between

1.3.156

segment

piece of cable to which devices are attached, terminated at both ends by its characteristic impedance. Segments may consist of several sections (non-terminated) connected by connectors

1.3.157

segmentation

division of a long message into several shorter frames for transmission

1.3.158

Send Queue

queue for sending Message Data in a device

1.3.159

service

capabilities and features of a sub-system (e.g. a communication layer) provided to a user

1.3.160

Session Header

part of a Message Data frame relevant to the Session Layer

1.3.161**Session Layer**

OSI layer in charge of establishing and closing communication

1.3.162**Side A**

one side of a vehicle with respect to a WTB Node

1.3.163**Side B**

other side of a vehicle with respect to a WTB Node

1.3.164**Slave**

device which receives information from the bus or sends information on the bus in response to a request (also called a poll) from the Master

1.3.165**Slave Frame**

frame sent by a Slave

1.3.166**Slave Start Delimiter**

Start Delimiter of a Slave Frame on the MVB

1.3.167**Source Device**

sender of a frame at the Link Layer (see: destination device)

1.3.168**sporadic transmission**

transmission which is made upon demand, when an event external to the network requires it (also called aperiodic, event-driven, demand-driven transmission)

1.3.169**Sporadic Data**

data frames transmitted on demand to carry Message Data or Supervisory Data

1.3.170**Sporadic Phase**

second half of a Basic Period, dedicated to the demand-driven transmission of messages and bus management data

1.3.171**star coupler**

device which takes the light of an optical fibre and redistributes it to several other fibres

1.3.172**Start Delimiter**

delimiter which announces the beginning of a frame on the MVB

1.3.173

Station

device capable of message communication, by contrast to simple devices, and which supports an Agent Function

1.3.174

Station Directory

directory which maps a Station Identifier to a Link Address and vice-versa

1.3.175

Station Identifier

8-bit identifier of a Station

1.3.176

Station Status Word

16-bit descriptor of the status and capabilities of a Station

1.3.177

Strong Master

Strong Node is currently Master and will not relinquish mastership until demoted to Weak Node status

1.3.178

Strong Node

Node selected by the application to become Strong Master. There may be only one Strong Master on a bus segment

1.3.179

stub

T-connection branching from an electrical bus line (at the tap), connecting a device to the line

1.3.180

Subscriber

one of the sinks of a broadcast Dataset (see: Publisher)

1.3.181

Supervisory Data

data transmitted within one bus only for the purpose of Link Layer supervision (e.g. arbitration on the MVB or Inauguration on the WTB)

1.3.182

System Address

Network Address of a Management Message exchanged between Manager and Agent, consisting of Node Address and Station Identifier

1.3.183

tap

place where a segment is tapped. A tap is a three-way electrical fork

1.3.184

Telegram

Master Frame and the corresponding Slave Frame, treated as a whole

1.3.185**terminator**

circuit which closes an electrical transmission line, ideally by its characteristic impedance

1.3.186**Terminator Switch**

switch which inserts the Terminator at the end of a segment on the WTB

1.3.187**Topography**

data structure describing the Nodes attached to the Train Bus, including their address, orientation, position and Node Descriptor

1.3.188**topology**

possible cable interconnection and number of devices a given network supports

1.3.189**Topography Counter**

counter in a Node which is incremented at each new Inauguration

1.3.190**Traffic Store**

shared memory accessed both by the network and the user, which contains the Process Data Port

1.3.191**Train Communication Network**

data communication network for connecting programmable electronic equipment on-board rail vehicles

1.3.192**Train Bus**

bus connecting the vehicles of a train, in particular, the WTB, and which conforms to the TCN protocols

1.3.193**Train Network Management**

services of the Network Management for TCN

1.3.194**transceiver**

combination of a transmitter and of a receiver

1.3.195**transmitter**

electronic device which can transmit a signal on the physical medium

1.3.196**Transport Data**

data carried by the Transport Layer, but not relevant to it

1.3.197**Transport Header**

part of a Message Data frame relevant to the Transport Layer

1.3.198

Transport Layer

layer of the OSI model responsible for end-to-end flow control and error recovery

1.3.199

trunk cable

cable which runs along the vehicles, as opposed to extension cable or jumper cable

1.3.200

User Address

Network Address of a User Message exchanged between Functions, consisting of Node Address (or Group Address) and Function Identifier

1.3.201

User Message

messages exchanged between user Functions

1.3.202

Variables

transmission service of the TCN

1.3.203

Var_Offset

bit offset of a Process Variable within a Dataset

1.3.204

Vehicle Bus

bus connecting equipment within a vehicle or group of not separable vehicles, in particular, the MVB, and which conforms to the TCN protocols

1.3.205

Vehicle Descriptor

application-dependent information about a particular vehicle, such as length and weight

1.3.206

Weak Master

Weak Node is currently Master and which will relinquish mastership if it finds another, stronger Master

1.3.207

Weak Node

Node which may take over the bus mastership spontaneously, but which releases it if it detects a stronger Node

1.3.208

Wire Train Bus (WTB)

Train Bus for frequently coupled and uncoupled vehicles, such as international UIC trains

1.4 Legends and abbreviations

ALI	Application Layer Interface, the definition of the semantics of all network services used by the application (a set of primitives, expressed as procedures, constants and data types)
AMA	Application Messages Adapter, the code directly called by the application which implements the Messages service
AMI	Application Messages Interface, the definition of the message services
ANSI	American National Standard Institute, a standardisation body in the United States
ASI	Application Supervision Interface, the definition of the Management services
ASN.1	Abstract Syntax Notation Number 1 on data presentation (ISO/IEC 8824)
AVA	Application Variables Adapter, the code directly called by the application implementing the Process Variable services
AVI	Application Variables Interface, the definition of the Process Variable services
BER	Basic Encoding Rules, a transfer syntax for ASN.1 data types (ISO/IEC 8825)
BR	Bit Rate, the rate of data throughput on the medium expressed in bits per second (bit/s) or in hertz (Hz), whichever is appropriate
BT	Bit Time, the duration of the transmission of one bit, expressed in μ s
ITU	International Telecommunication Union, the international standardisation body for telecommunications based in Geneva
CRC	Cyclic Redundancy Check, a data integrity check based on polynomial division
DIN	Deutsches Institut für Normung, the German national standardisation body
EIA	Electronics Industries Association, a standardisation body in the United States
EMD	Electrical Middle Distance, one of the media of the MVB
ESD	Electrical Short Distance, one of the media of the MVB
EP	Electro-Pneumatic brake cable as described in UIC leaflet 648
ERRI	European Railways Research Institute, laboratory based in Utrecht, Netherlands
FCS	Frame Check Sequence, an error detection code appended to the transmitted data, as specified in ISO/IEC 3309
GCT	Guidelines for Conformance Test, the conformance testing specified in Annex B of this Standard
HDLC	High-level Data Link Control, a Link Layer protocol whose frame format is defined in ISO/IEC 3309
IEC	International Electrotechnical Commission, Geneva
IEEE	Institute of Electrical and Electronics Engineers, New York

ISO	International Standard Organisation, Geneva
LLC	Logical Link Control, a sub-layer within the Link Layer ruling the data exchange
LME	Layer Management Entity, the entity in charge of supervising a layer on behalf of Network Management
LMI	Layer Management Interface, the services provided by the LME
MAC	Medium Access Control, a sub-layer within the Link Layer ruling which device is entitled to send on the bus
MAU	Medium Attachment Unit, the part of a Node which interfaces electrically to the bus and which provides/accepts binary logic signals
MIB	Management Information Base, the set of all objects accessed by Network Management
MVB	Multifunction Vehicle Bus, a Vehicle Bus
NRZ	Non-Return to Zero, the simplest encoding scheme in which one bit is represented by one level for a "1" and the other level for a "0", or vice-versa, with a separate clocking
OGF	Optical Glass Fibre, one of the media of the MVB
ORE	Office de Recherches et d'Essais, a UIC laboratory based in Utrecht, Netherlands
OSI	Open System Interconnection, a universal communication model defined in the ISO/IEC 7498
PCTR	Protocol Conformance Test Report, defined in ISO/IEC 9646
PICS	Protocol Implementation Conformance Statement, defined in ISO/IEC 9646
PTA	Process Data to Traffic Store Adapter, the component which accesses one of the Traffic Stores
RIC	Regulation for the reciprocal use of coaches and vans in international traffic, edited by UIC.
RTP	Real-Time Protocols, the common communication protocols given in Clause 2 of this standard
SDL	Specification and Description Language, a specification language defined by ITU-T Z100 Annex D for communication protocols
TCN	Train Communication Network, a set of communicating vehicle and Train Busses
TNM	Train Network Management
UIC	International Union of Railways , the international railways operators association
WTB	Wire Train Bus

1.5 Conventions

1.5.1 Base of Numeric Values

This Standard uses a decimal representation for all numeric values unless otherwise noted.

Analog and fractional values include a coma.

EXAMPLE - The voltage is 20,0 V.

Binary and hexadecimal values are represented using the ASN.1 (ISO/IEC 8824) convention.

EXAMPLE - Decimal 20 coded on 8 bits = '0001 0100'B = '14'H

1.5.2 Naming conventions

Keywords in the TCN specifications are written with a capital letter at the beginning.

If the name is composed, the different parts of the name are united with a space.

When a data structure is associated with a keyword, its type consists of the same basic names separated by an underscore.

When the value corresponding to the keyword is transmitted in a message, the corresponding field has the same name as the type, but in lower case.

When the value is passed as parameter, the parameter name has the same name as the field in a message.

In the SDL diagrams, the corresponding variable has the same name as the type, but without underscores.

EXAMPLES

Topo Counter is a counter of the link layer;

It is of the type Topo Counter, which is an UNSIGNED6.

When its value is transmitted in a message, the corresponding field is called 'topo_counter'.

When its value is transmitted across a procedural interface, the parameter is called 'topo_counter', its C-type is Type_Topo_Counter.

In the SDL diagrams, the variable representing the counter is called TopoCounter.

1.5.3 Time naming conventions

Time values beginning with a lower case (e.g. t mm) are measurable time intervals.

Time values beginning with an uppercase (e.g. T reply) are parameters or time-out values.

1.5.4 Procedural interface conventions

A procedural interface is defined by a set of service primitives, which represent an abstract, implementation-independent interaction between the service user and the service provider.

These primitives are expressed in this Standard as procedures in the ANSI C syntax with typed parameters.

This ought to be considered as a semantic description only, which does not imply a particular implementation or language. Any interface which provides the same semantics is allowed.

Conformance to the syntax of this interface cannot be claimed. Implementations are free to change the procedure or parameters names, to add parameters or to split procedures, as long as the specified service is provided.

Interface procedures are defined in the ANSI C syntax using the Courier font.

Procedure names, variables and parameters appear in all lower case.

EXAMPLE - `lm_send_request`

Constants and type definitions appear in all upper case.

EXAMPLE - `UNSIGNED32`

The name of a procedure or of a type is prefixed:

for the Variables service by:

- `lp_` or `LP_` Link Layer
`ap_` or `AP_` Application Layer

for the Messages service by:

- `MD_` messages in general
`lm_` or `LM_` Link Layer
`nm_` or `NM_` Network Layer
`tm_` or `TM_` Transport Layer
`sm_` or `SM_` Session Layer
`am_` or `AM_` Application Layer

Table 1 shows a template used for procedures and types.

Table 1 - Template for the specification of an interface procedure

Definition	<i>The service or data type is expressed here.</i> <i>In case of an indication procedure, the event which triggers the call is indicated here, beginning with "When"</i> <i>The name and parameters of the service procedure are defined here.</i> <i>In case of an indication procedure, the type of the procedure is specified.</i> <i>Input parameter, output parameters and return parameter are distinguished.</i>	
Syntax	MD_RESULT lm_send_request (/* example */ destination, link_control, p_packet status) unsigned, UNSIGNED8 MD_PACKET * ENUM8 *	
Input	<i>Input parameters are supplied to the procedure, which is not allowed to modify them.</i>	
	destination	The data type "unsigned" is compiler-dependent.
	link_control	parameter passed by reference, not modified by the procedure. The data type is an 8-bit word.
	p_packet	The "*" denotes a pointer to the p_packet data structure, which is of type MD_PACKET, defined elsewhere in this Standard.
Output	<i>Output parameters are expected to be modified by the call.</i>	
	status	The type ENUM8 is an 8-bit enumeration type.
Result	<i>The Result parameter is an optional output parameter which expresses success or failure of the call, but not necessarily of the service.</i>	
	MD_RESULT	The Result parameter is of the type: AM_RESULT for the AMI, MD_RESULT for the LMI, LP_RESULT for the LPI, AP_RESULT for the AVI, The template specifies the error codes expected for each procedure individually. The Result parameter is not explicitly described if the only two values expected are: xx_OK = 0 successful completion; xx_FAILURE <> 0 some problem. The result can also be returned as an output parameter in the parameter list, depending on the implementation.
Usage	The rules listed after the procedure template indicate how the procedure should be used. Although usage rules are not mandatory, not following them produces unpredictable results.	

NOTE Data structures represented in this table are interface specifications which should not be confused with formats of the same data structures when transmitted over a bus, see 1.5.5.

1.5.5 Specification of transmitted data

The format of transmitted data, single frames as well as whole messages, is specified in two forms:

- a graphical form, which is not normative, but shows the message structure at a glance.
- a textual form based on ASN.1, with encoding rules specified in 2.3.

EXAMPLE - A graphical form of a message is shown in Table 2, the corresponding textual form is shown in Table 3.

Table 2 - Example of message structure

first transmitted octet								next transmitted octet								
bit->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	snu	gni	node_id						station_id							
2	next_station_id								rsv1	Tvd	topo_counter					
4	tnm_key								str_code (= 3)							
6	parameter 1															
8	parameter 2								parameter 3						par4	
parameter 5: ARRAY [n] OF (repeat following field (n) times)																
parameter 5.1																
parameter 5.2																
parameter 5.3: STRING32																
(CHARACTER8)								last character or 0								

^octet

In the graphical form, one line is used for each word of 16 bits, but in Clause 4 (WTB), lines are 8-bit oriented.

Arrays of parameters are preceded by a repetition frame on the top and to its left.

Repetitions can be nested (see parameter 5.3 in Table 4)

If the size of a parameter may be longer than three words, three lines are allocated for it and the middle one has a shaded border.

Table 3 - Example of textual message form (corresponding to table 2)

Call_Mgt_Message ::= RECORD

```

{
  snu                BOOLEAN1 (=1)      -- this '1' means that message
                                         uses system addressing
  gni                BOOLEAN1 (=0)      -- this '0' means that final is
                                         an individual device
  node_id            UNSIGNED6          -- 6-bit node address of final
  station_id         UNSIGNED8          -- 8-bit identifier of the
                                         station
  next_station_id    UNSIGNED8          -- 8-bit identifier of next
                                         station
  rsv1              BOOLEAN1 (=0)      -- this bit is always 0
  tvd               BOOLEAN1          -- this bit indicates if
                                         topography counter is valid
  topo_counter       UNSIGNED6          -- 6-bit topography counter
  tnm_key            UNSIGNED8,         -- announces a network
                                         management call message.
  sif_code           UNSIGNED8,         -- there is a different SIF_code
                                         for each Management Message
  parameter1         INTEGER16,        -- a 16-bit value. If the
                                         parameter has less than 16
                                         bits, the value is right-
                                         justified and sign-extended
                                         (e.g. one single octet is
                                         transmitted as the second
                                         octet).
  parameter2         INTEGER8,         -- this value is transmitted in
                                         the most significant part of
                                         a word.
  parameter3         UNSIGNED6         -- this value is transmitted in
                                         the least significant octet,
                                         but the lower two bits of
                                         that octet are reserved for
                                         parameter4.
  par4              ANTIVALENT2        -- parameter4 has two bits
  parameter5         ARRAY [n] OF     -- parameter5 represents a
                                         structured data to be
                                         {
                                         -- repeated n times, containing:
                                         parameter5.1    INTEGER16      -- first parameter of the
                                         -- repeated field
                                         parameter5.2    UNIPOLAR4.16    -- second parameter of the
                                         -- repeated field
                                         parameter5.3    STRING32        -- third parameter is a string
                                         -- (array of up to 32 8-bit
                                         -- characters);
                                         -- a string is closed by a
                                         -- '0' character, or by two
                                         -- such '0' characters to align
                                         -- on a 16-bit word boundary;
                                         -- a void string consists of
                                         -- 32 '0' characters;
                                         -- the actual size of a
                                         -- string is deduced from the
                                         -- number of significant
                                         -- characters before the zero.
                                         },
}

```

Fields names begin with a lower case letter, their type begins with an upper case. Sometimes, the same type is used as a transmission format, in which case only the first letter is upper case, and as a C-type, in which case the whole type is in upper case.

EXAMPLE: Am_Result (transmission format) and AM_RESULT (C-type of an interface procedure).

1.5.6 State diagram conventions

The transport protocol state machine is described as in ISO/IEC 8802.2 (Logical Link Layer) in the form of a table, which specifies the transitions between the possible states in which a state machine may be.

Transitions between states are governed by events, coming from the Network Layer (inbound packets), from the Session Layer (commands) or from time-outs.

An action depending on the event is executed before leaving the state. This action defines the next state.

Figure 2 shows an example of a state transition diagram.

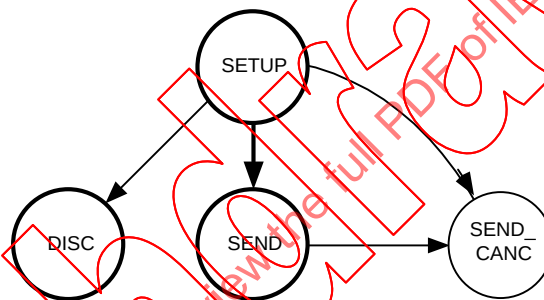


Figure 2 - State transition example

From “SETUP”, the machine may go to three different states, DISC, SEND or SEND_CANC.

The transition between these states are governed as Table 4 shows.

Table 4 - State transitions table

Current State	Event	Action(s)	Next state (if <> current)
SETUP	rcv_DR	close_send (DR_reason);	DISC
	rcv_CC AND (conn_ref = CC_conn_ref)	IF (eot) THEN close_send (AM_OK); ELSE credit := CC_credit; send_not_yet := credit; send_data_or_cancel; END;	DISC SEND or SEND_CANC
	TMO AND (rep_cnt = MAX_REP_CNT)	close_send (AM_CONN_TMO_ERR);	DISC

According to Table 4, there are three events which cause a transition from SETUP to the DISC state:

- 1) rcv_DR (received Disconnect_Request, a network event). The corresponding action consists in closing the connection (close_send) before going to state DISC;
- 2) another network event (received Connect_Confirm with correct reference), leading either to state DISC, SEND or SEND_CANCEL, depending on the outcome of the send_data or cancel procedure;
- 3) A time-out conditioned by the predicate (rep_cnt = MAX_REP_CNT), which also causes the closing of the connection.

1.6 General considerations

1.6.1 Interface between equipment

This Standard defines the data communication interface of equipment located in a vehicle as a connection of devices to a Vehicle Bus, as shown in Figure 3.

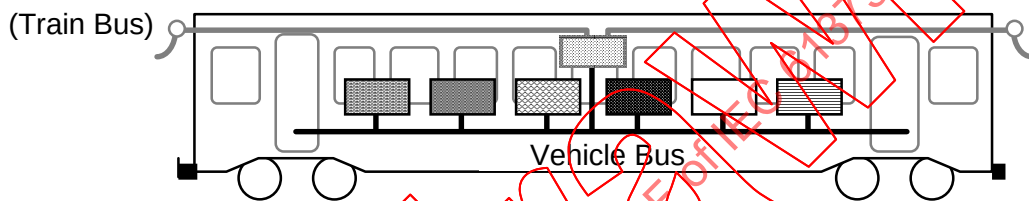


Figure 3 - Interfaces between equipment

As Vehicle Bus, Clause 3 of this Standard specifies the Multifunction Vehicle Bus (MVB), a serial data communication bus designed primarily, but not exclusively, for interconnecting equipment where interoperability and interchangeability is needed.

1.6.2 Interface between vehicles

This Standard defines the data communication interface between vehicles as the connection of Nodes located in vehicles to a Train Bus, as shown in Figure 4.

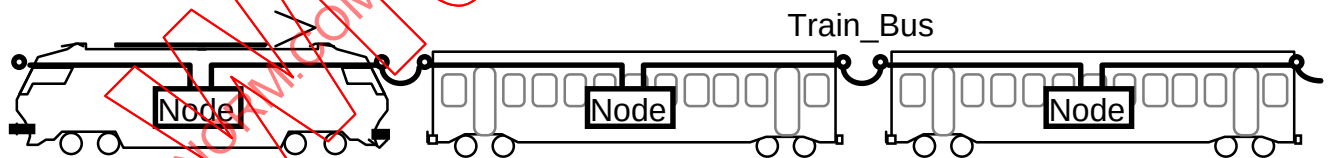


Figure 4 - Interfaces between vehicles

As Train Bus, Clause 4 of this Standard specifies the Wire Train Bus (WTB), a serial data communication bus designed primarily, but not exclusively, for interconnecting vehicles of Open Trains.

NOTE For a definition of Open Trains, see 1.3.122

1.6.3 Real-Time Protocols

This Standard defines the architecture of the TCN as a hierarchy consisting of two levels, a Train Bus and a Vehicle Bus, as shown in Figure 5.

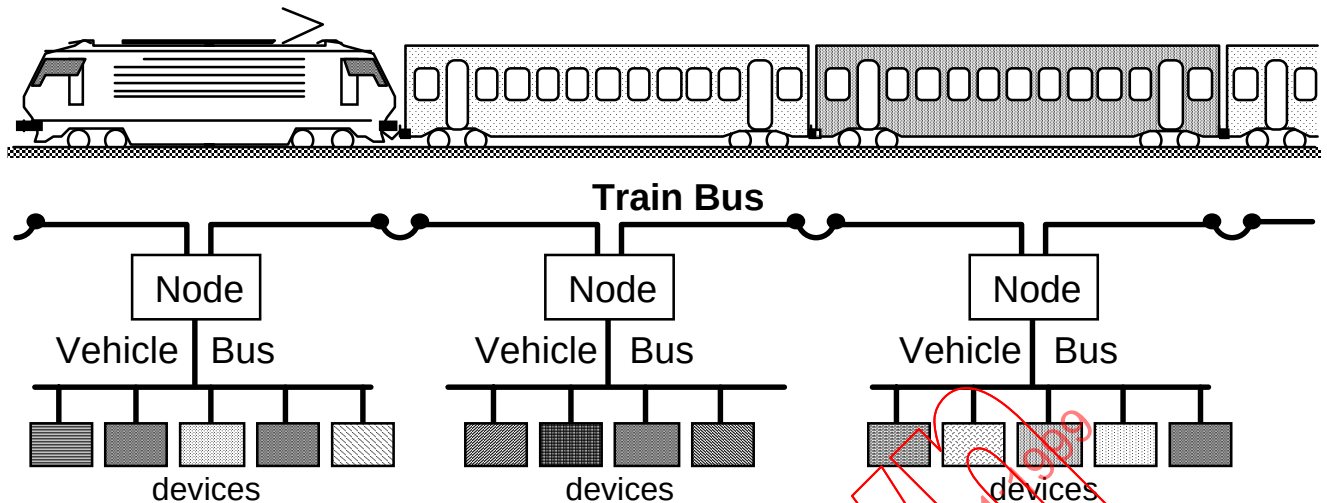


Figure 5 - Train Bus and Vehicle Bus

As communication protocols, Clause 2 of this Standard specifies the Real-Time Protocols (RTP), used by all devices in the TCN to communicate over the MVB, the WTB or any other bus.

The RTP specify the Application Interface which the TCN provides, consisting of two basic services: Variables and Messages.

The RTP specify the transmission protocols which handle in particular routing, flow control and error recovery.

The RTP specify the interface which the busses are expected to provide to the transmission protocols, and in particular the two basic services:

- a) cyclic, source-addressed broadcast of Process Data; and
- b) sporadic, connectionless transmission of Message Data.

1.6.4 Network Management

As Network Management, Clause 5 of this Standard specifies the TCN Network Management (TNM) as a collection of messages exchanged between Manager and Agent to provide the basic services.

1.6.5 Configurations

This Standard can be used in parts or as a whole. For instance, it is possible to use:

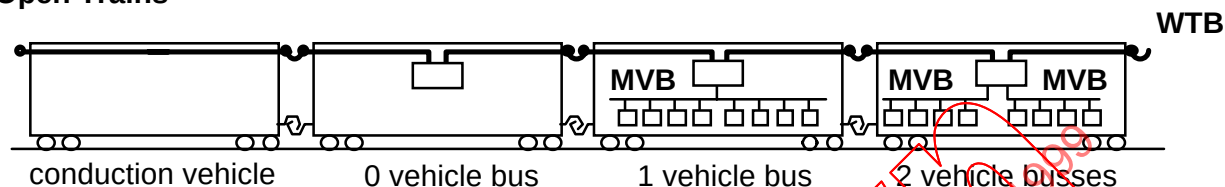
- a) the WTB without a vehicle bus or with another vehicle bus than MVB;
- b) the MVB without a train bus or with another Train Bus than the WTB;
- c) the RTP with other busses than the WTB or the MVB.

Figure 6 shows three configurations corresponding to three application domains:

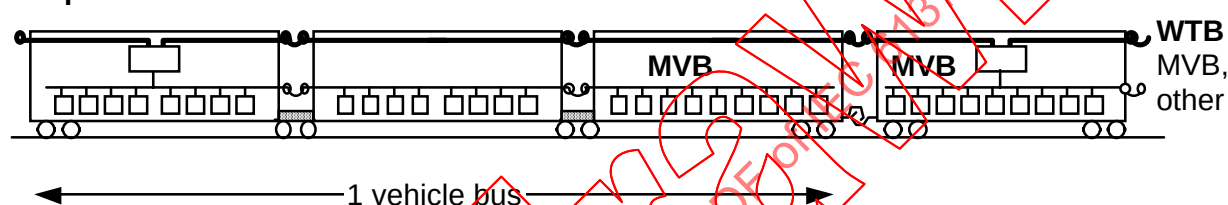
- a) the Open Trains configuration shows an Open Train, such as a UIC train, which require automatic configuration. The WTB is used as the standard Train Bus, supporting up to 32 Nodes. There may be 0, 1 or more Nodes per vehicle. To each Node, up to 15 Vehicle Busses (MVB or others) may be attached.

- b) the Multiple Unit Train configuration shows two connected closed trains. When these closed trains are coupled and uncoupled frequently, the WTB can be used as a standard Train Bus, but when configuration by other means is possible, other busses such as the MVB can be used instead. The Vehicle Bus can span several vehicles.
- c) the Closed Trains configuration shows a closed train, in which the MVB can be used both as Train Bus and as Vehicle Bus.

Open Trains



Multiple Unit Trains



Closed Trains

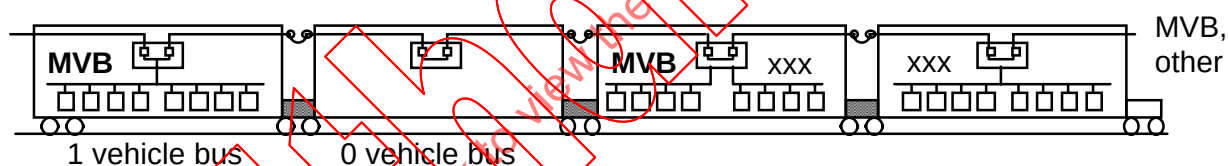


Figure 6 - TCN configurations

In all three configurations of Figure 6, the MVB is used to connect on-board equipment, but other busses may be used as Vehicle Bus.

1.6.6 Structure of a standard device

This Subclause specifies permitted options in a TCN device. The options are described in the corresponding Clauses and summarised in Figure 7.

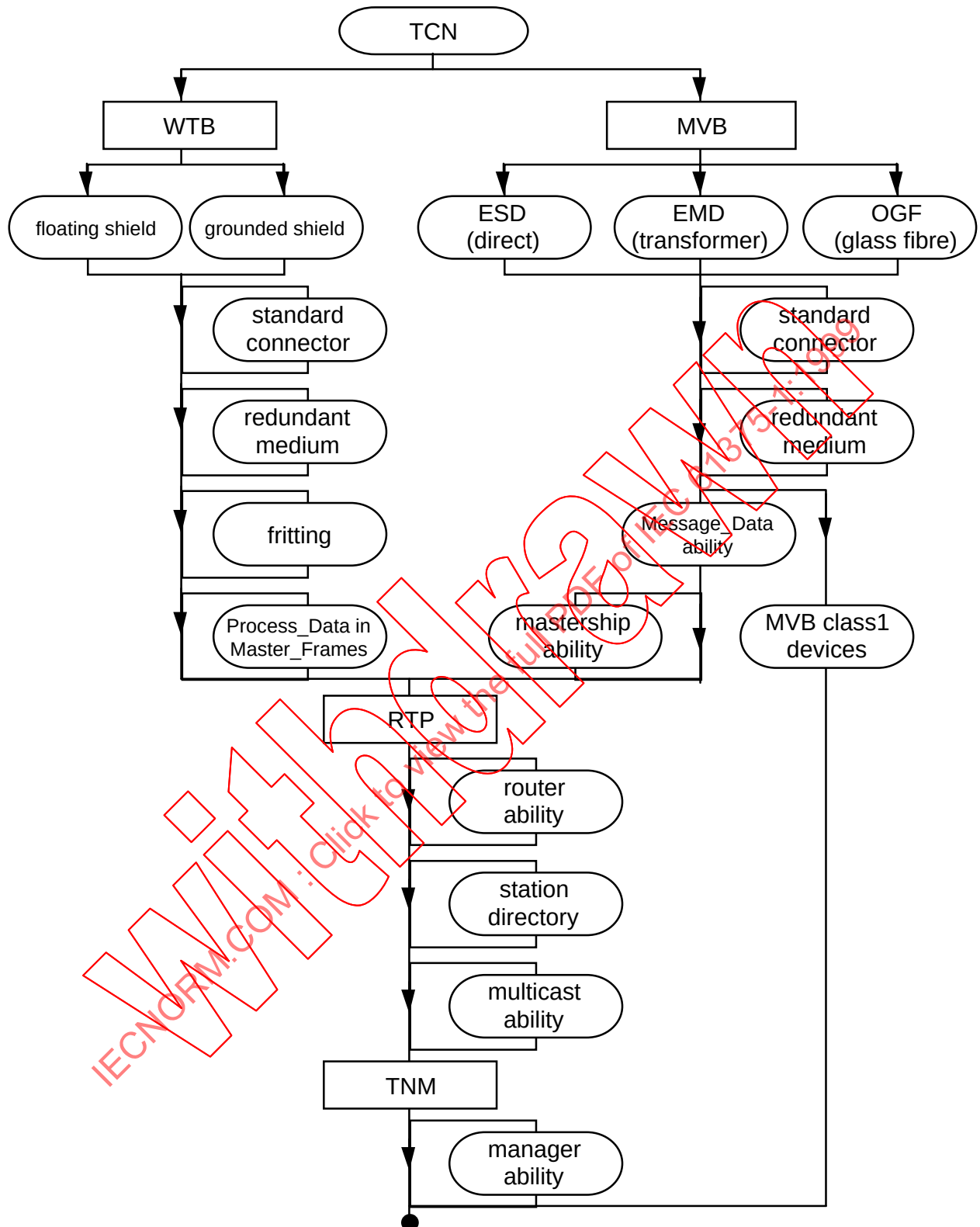


Figure 7 - TCN device configuration options

1.6.6.1 TCN device

A TCN compliant device shall implement at least one bus MAU for the MVB or for the WTB.

1.6.6.2 MVB options

The total number of TCN bus MAUs in a device shall not exceed 16.

An MVB MAU shall use one of the three standard media: ESD, EMD or OGF.

An MVB MAU may use a connector as specified in this Standard.

An MVB MAU may use a redundant medium as specified in this Standard.

An MVB MAU may include the Message Data transmission ability.

An MVB MAU may include the mastership ability.

1.6.6.3 WTB options

A WTB MAU shall be configurable for either a floating shield or a grounded shield.

A WTB MAU may use a connector as specified in this Standard.

A WTB MAU may use a redundant medium as specified in this Standard.

A WTB MAU may implement frtting as specified in this Standard.

A WTB MAU may transfer Process Data in Master Frames as specified in this Standard.

1.6.6.4 RTP options

A TCN device shall implement the Variables services.

A TCN device, except for MVB Class 1 device, shall implement the Messages services.

A TCN device may implement the router function if it has more than one MAU.

A TCN node may implement the node directory.

A TCN device may implement the station directory.

A TCN device may implement the multicast protocol.

1.6.6.5 TNM options

A TCN device shall implement the Agent function.

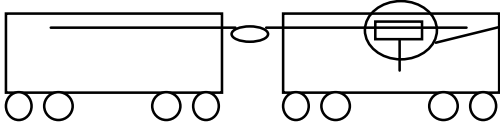
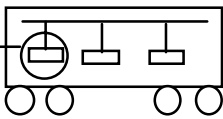
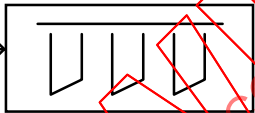
A TCN device may implement the Manager function.

1.7 Conformance Test

To claim conformance to the TCN standard, devices are expected to pass a suite of tests.

To ensure that interoperability is guaranteed according to table 5, this Standard comprises a set of Guidelines for Conformance Test, which are listed in Annex B.

Table 5 - Interoperability Testing

	COVERED BY	
	IEC 61375	End USER
Level 1 Interface between vehicles (Train BUS) 	PROTOCOL PHYSIC/ELEC Node connector cable specification (Z: attenuation)	USER MESSAGES MECHANICAL (connectors/cables)
Level 2 Interface between equipment (Vehicle BUS) 	PROTOCOL PHYSIC/ELEC MECHANICAL (connectors/cables)	USER MESSAGES
Level 3 Interface between boards or Inter components 	Not covered	Not covered

2 Real-Time protocols

This clause applies to a TCN which uses WTB (clause 4) and/or MVB (clause 3) and/or any other bus which obeys to the same operating principles.

A tutorial to the RTP is given in A.2.

2.1 General

2.1.1 Contents of this clause

This clause specifies one component of the Train Communication Network, the Real-Time Protocols, which provides communication between applications, within vehicles and between vehicles, as illustrated on figure 8.

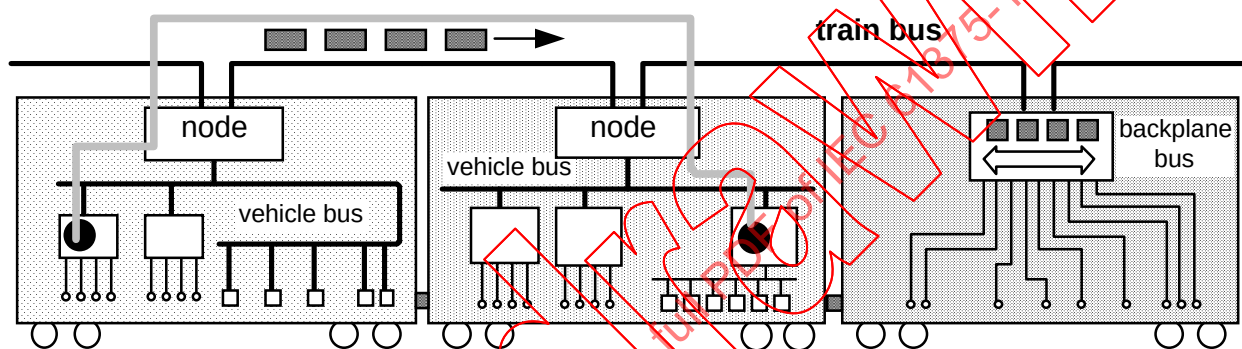


Figure 8 - Structure of the Train Communication Network

This Clause specifies the two main communication services to the application:

- a) Variables: transfer of short data with deterministic delivery delay, including
 - Link layer Interface for Process_Data (LPI);
 - Application layer Interface for Variables (AVI);
- b) Messages: transmission of possibly lengthy, but infrequent data items divided if necessary into small packets and transmitted on demand, including:
 - Link layer Interface for Message_Data (LMI);
 - Network layer routing used for routing packets in the network;
 - Transport layer which provides flow control and error recovery,
 - for point-to-point or
 - for multicast messages (optional),
 - the Session layer which pairs Call_Message and Reply_Message;
 - the Application layer Interface for Messages (AMI).

This Clause also specifies the data presentation (for Variables and for Messages).

2.1.2 Structure of this clause

This clause is structured similarly to the OSI communication model as shown in Figure 9.

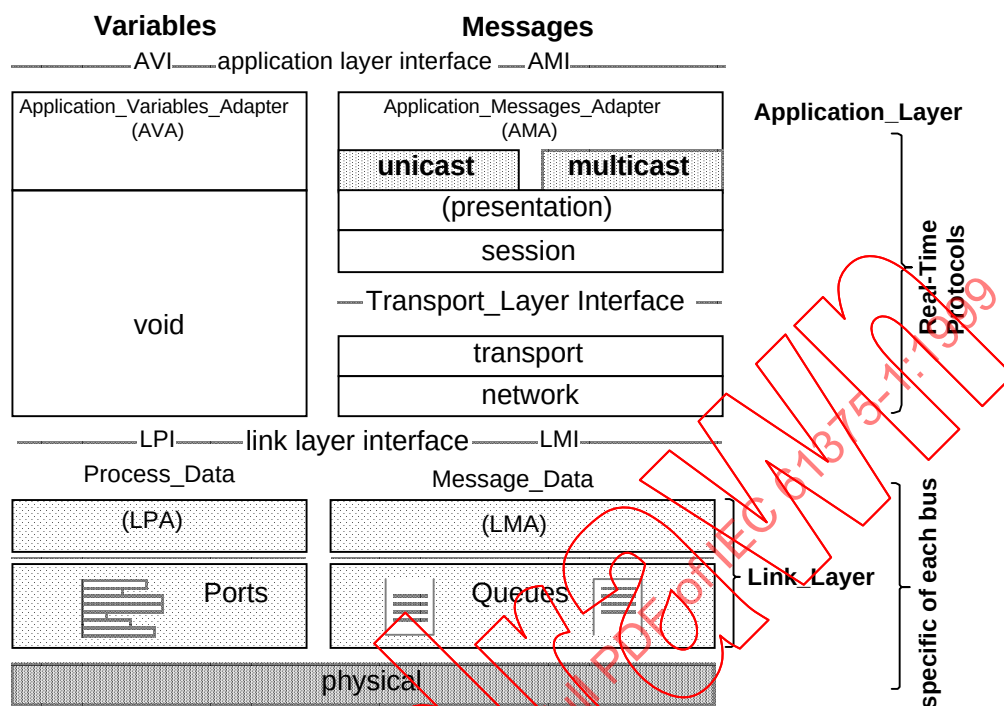


Figure 9 - Real-Time Protocols layering

Subclause 2.1 General

Normative requirements and definitions

Subclause 2.2 Variables - Services and Protocols

Process_Variable Objects

Dataset Interface

Application interface for individual, Set and Cluster access

Subclause 2.3 Messages Services and Protocols

Architecture

Link layer Interface

Network layer

Transport layer

Session layer

Application Interface

Subclause 2.4 Presentation and encoding of transmitted and stored data

2.2 Variables - Services and Protocols

2.2.1 General

For a tutorial on Process_Variables, see subclause A.2.2.

The services and protocols are separated into a lower interface and a higher interface:

- a) the lower, link layer interface, which specifies the services expected from the bus; and
- b) the application layer interface specifies the services given to the application.

2.2.2 Link layer Interface for Process_Data

2.2.2.1 Purpose

The Link_Process_Data_Interface (LPI) defines the Process_Data services provided by a bus to the higher protocols.

The LPI defines the initialisation of the ports, the inclusion and the removal of whole Datasets into the ports and the synchronisation primitives associated with the transmission of whole Datasets.

An Application normally does not access the LPI directly, except for synchronisation (to be informed of reception or transmission of Datasets).

The underlying communication is not specified by this interface. The transmission between ports, including the polling strategy of the bus master, is realized by the link layer and the physical layer, as described for instance in the MVB (Clause 3) or WTB (Clause 4).

NOTE Individual Process_Variables are not visible at the LPI level.

2.2.2.2 Datasets

2.2.2.2.1 Ports and Traffic_Store

The link layer shall provide a number of ports for Process_Data communication.

A port is a shared memory structure, which can be accessed simultaneously by the application and the network.

A port is a non-queued data structure, meaning that its contents are overwritten by a new written value and are not affected by a read operation.

The link layer as well as the application shall be able to access a port consistently, i.e. write or read all its data in one indivisible operation.

Ports belonging to the same link layer belong to the same Traffic_Store.

A port shall be identified within a Traffic_Store by its Port_Address.

A Traffic_Store shall be identified within a device by its Traffic_Store_Id.

2.2.2.2.2 Dataset consistency

Each port shall contain exactly one Dataset.

A Dataset shall be produced by only one Publisher application.

There shall be only one source port with a given Port_Address on a bus, but there may be an undefined number of sink ports.

The link layers of the different devices shall transmit the contents of a source port within a limited time to the sink ports subscribed to the same Port_Address and provide consistency of transmitted Datasets.

NOTE The bus is not expected to guarantee that different Datasets can be transmitted or retrieved as a consistent set.

2.2.2.2.3 Error handling

Undefined fields in a Dataset shall be overwritten with '1'.

If the link layer cannot guarantee consistency of a Dataset, e.g. if it detects that a transmission error occurred or that its Publisher application is not able to supply correct or timely data, the link layer shall overwrite the whole port with '0'.

NOTE Since overwriting the value of a Process_Variable with all '0' or '1' may yield a legal value, a Check_Variable of the same Dataset is used as a validity indicator where this could be a problem.

2.2.2.2.4 Freshness supervision

Each sink port and therefore each subscribed Dataset shall have an associated Freshness_Timer, which indicates the time elapsed since the bus wrote a new value to this port.

This Freshness_Timer shall be retrieved in an indivisible operation with the Dataset contents.

The resolution of Freshness_Timer shall be shorter or equal to 16 ms.

It shall have a range of at least 4 s and stop when reaching end-of-range.

NOTE 1 The Freshness_Timer does not consider when the Publisher Application inserted the Process_Variable into the port. Source time supervision is an application issue which can be handled by Check_Variables.

NOTE 2 Freshness_Timer is independent from a possible forcing of Variables.

2.2.2.2.5 Synchronisation Dataset

The broadcast transmission of given Datasets may be used to synchronise applications.

NOTE On the MVB, a Dataset is dedicated to time transmission.

2.2.2.2.6 Dataset polling

The procedures which concern Dataset polling are part of the link layer respectively of the vehicle bus and of the train bus. They are not described in this standard.

2.2.2.2.7 Dataset identifier (DS_Name)

2.2.2.2.7.1 Dataset, port and Logical Address

Within a device, a Dataset shall be identified its Traffic Store and by the Port_Address of the Traffic_Store where the Dataset is stored.

When transmitted over a bus, a Dataset shall be identified by the Logical_Address of its Process_Data frame on that bus.

The Logical_Address of the Process_Data frame shall be identical to the Port_Address of the Traffic_Store where the Dataset is stored.

2.2.2.2.7.2 DS_Name format

A Dataset shall be identified within a device by its DS_Name.

Definition	Type of Dataset
Syntax	<pre>typedef struct /* big-endian representation */ { unsigned traffic_store_id :4, /* DS_NAME first part */ unsigned port_address :12, /* DS_NAME second part */ } DS_NAME;</pre>

The DS_Name can be conveniently represented by a 16-bit word, as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Traffic_store_id				port_address											

2.2.2.2.7.3 Traffic_Store Identifier

Traffic_Store_Id shall select one of the Traffic_Stores in a device.

The maximum number of Traffic_Stores supported shall be sixteen.

NOTE 1 The Traffic_Store identifier does not imply which type of bus (MVB, WTB or other) is accessed, but an implementation can be simplified if it does (e.g. a WTB Traffic_Store could be always '1').

NOTE 2 The Traffic_Store identifier may be identical to the Bus_Id of the corresponding bus. There may, however, exist Traffic_Stores without a connected bus, for instance for inter-task communication.

2.2.2.2.8 Port Address

The Port_Address shall identify one of 4096 ports within the Traffic_Store selected by Traffic_Store_Id.

NOTE The actual possible number of ports depends on the kind of bus connected.

EXAMPLE - On the MVB, there may be up to 4096 ports of up to 256 bits each per device.

2.2.2.3 Link_Process_Data_Interface primitives

2.2.2.3.1 General

The Link_Process_Data_Interface (LPI) shall provide for Dataset access the primitives illustrated in Figure 10 and listed in Table 6, as specified in the following subclauses.

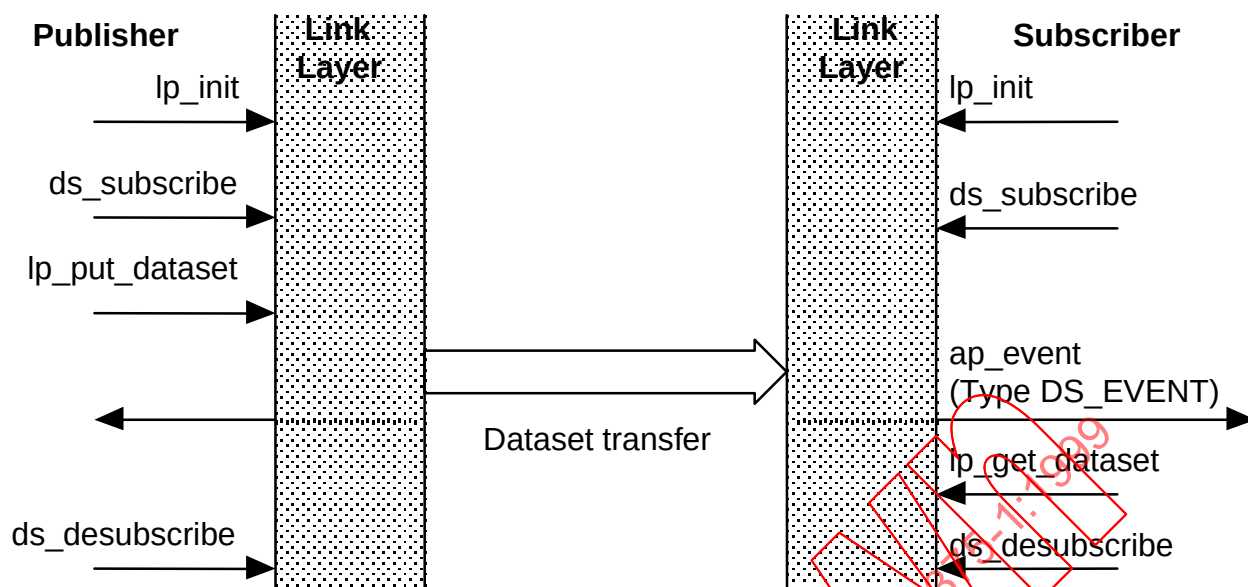


Figure 10 - LPI primitives exchange

Table 6 - LPI primitives

Name	Meaning
<code>lp_init</code>	Initialises the Traffic_Store
<code>lp_put_dataset,</code>	Inserts a Dataset to send
<code>lp_get_dataset.</code>	gets a received Dataset
<code>ds_subscribe,</code>	Subscribes a Dataset for synchronisation
<code>ap_event</code>	Synchronisation upon transmission or reception
<code>ds_desubscribe,</code>	Unsubscribes a Dataset from synchronisation

NOTE 1 The Application can access directly the Traffic_Store structures rather than use these primitives to speed up access.

NOTE 2 A communication processor can use the same primitives to access the Traffic_Store from the bus side. On the MVB, it is expected that these primitives are executed in hardware.

NOTE 3 These primitives do not trigger immediately a communication over the bus, but access only the Traffic_Store.

2.2.2.3.2 Type 'LP_RESULT'

Definition	A procedure of the LPI which returns a value of type LP_RESULT shall encode it as follows:
Syntax	<pre>typedef enum { LP_OK = 0, /* correct termination */ LP_PRT_PASSIVE = 1, /* warning: Dataset not active */ LP_ERROR = 2, /* unspecified error */ LP_CONFIG = 3, /* configuration error */ LP_MEMORY = 4, /* not enough memory */ LP_UNKNOWN_TS = 5, /* unknown Traffic_Store */ LP_RANGE = 6, /* memory address error */ LP_DATA_TYPE = 7, /* unsupported data type */ } LP_RESULT;</pre>

2.2.2.3.3 Procedure 'lp_init'

Definition	Creates a Traffic_Store, sets up the subscription list, creates source and sink ports and initialises them to a defined value.	
Syntax	<pre>LP_RESULT lp_init (ENUM8 ts_id void * p_descriptor);</pre>	
Input	ts_id	Traffic_Store_Id (0..15)
	p_descriptor	Implementation-dependent data structure.
Return	any LP_RESULT	

2.2.2.3.4 Procedure 'lp_put_dataset'

Definition	Copies a Dataset from the Application to a port in the Traffic_Store.	
Syntax	<pre>LP_RESULT lp_put_dataset (DS_NAME * dataset; void * p_value);</pre>	
Input	dataset	DS_Name of the Dataset to be published.
	p_value	pointer to a memory region of the Application where the Dataset value is copied from.
Return	any LP_RESULT	
Usage	The previous value of the Dataset in the Traffic_Store is overwritten.	

2.2.2.3.5 Procedure 'lp_get_dataset'

Definition	Copies a Dataset and its Freshness_Timer from a port to the Application.	
Syntax	<pre> LP_RESULT lp_get_dataset (DS_NAME * dataset; void * p_value; void * p_fresh); </pre>	
Input	dataset	DS_Name of the Dataset to be received.
Return	Any LP_RESULT	
Output	p_value	Pointer to a Memory_Address of the Application where the Dataset value is copied to.
	p_fresh	Pointer to a Memory_Address of the Application where the Freshness_Timer is copied to.

2.2.2.3.6 Procedure 'ds_subscribe'

Definition	Subscribes a Dataset for transmission or reception and indicates which indication procedure is called in case the specified Dataset is transmitted or received.	
Syntax	<pre> LP_RESULT Ds_subscribe (DS_NAME * dataset; DS_EVENT event_cnf; UNSIGNED16 instance); </pre>	
Input	dataset	DS_Name of the Dataset to be included into the subscription.
	event_cnf	Subscribed procedure
	instance	16-bit reference number which identifies the subscribing Application instance and will be returned in the ds_event procedure.
Return	Any LP_RESULT	
Usage	1 – This procedure may be called several times up to the limits set by the implementation, for different Datasets and subscribed procedures. 2 – The same Dataset may be subscribed only once.	

2.2.2.3.7 Type 'DS_EVENT'

Definition	When a Dataset has been sent or received, the link layer shall call the procedure subscribed to that Dataset, and which shall be of type 'DS_EVENT'.	
Syntax	<pre>typedef void (* DS_EVENT) (UNSIGNED16 instance);</pre>	
Input	instance	16-bit reference number which identifies the Application instance which subscribed this event.
Usage	1 - This procedure has been subscribed previously by 'ds_subscribe'. 2 - The Dataset causing the event is not identified, but the instance parameter can be used for this purpose.	

2.2.2.3.8 Procedure 'ds_desubscribe'

Definition	Removes a Dataset from the subscription.	
Syntax	<pre>LP_RESULT Ds_desubscribe (DS_NAME * dataset;);</pre>	
Input	dataset	DS_Name of the Dataset to be removed from subscription
Return	Any LP_RESULT	

2.2.3 Application interface for Process_Variables**2.2.3.1 Purpose**

The Application_Variables_Interface (AVI) defines the Variables transfer services offered to the Application.

The primitives of this interface access only the ports in the Traffic_Store(s) and do not trigger a communication over the bus.

It is assumed that the inclusion of a Variable into a port by a Publisher will cause, after a limited time, the inclusion of the same Variable into the corresponding port of the Subscriber(s).

2.2.3.2 Process_Variables**2.2.3.2.1 Process_Variable transmission and storage**

Process_Variables are transmitted as part of a Dataset.

All Process_Variables belonging to a Dataset shall be transmitted and stored as a consistent set.

2.2.3.2.2 Freshness supervision

A Process_Variable shall be retrieved with the Freshness_Timer of its Dataset in an indivisible operation.

2.2.3.2.3 Synchronisation

An application may be synchronised by the transmission of a Dataset, over the LPI.

2.2.3.2.4 Check_Variable

To assess its validity, each Variable may be associated with another Variable pertaining to the same Dataset, called the Check_Variable.

The Check_Variable and the Process_Variable shall be stored and retrieved in an indivisible operation.

The same Check_Variable may apply to several Process_Variables.

The Check_Variable may be located at any place in the Dataset and may overlap a Process_Variable.

The Check_Variable, if used, shall have the format of an ANTI VALENT2 and take the following values:

- a) '00'B: the protected variables are erroneous or suspicious;
- b) '01'B: the protected variables are assumed to be correct;
- c) '10'B: the protected variables have been forced to an imposed value;
- d) '11'B: the protected variables are undefined

NOTE 1 The word 'Variable' will be used when it is not specified whether it is a Process_Variable or a Check_Variable.

NOTE 2 The allocation of Process_Variables and Check_Variables in the Dataset is an application issue.

NOTE 3 The bus or the Publisher are expected to overwrite suspicious fields in a Dataset with '0'. This sets both bits of the Check_Variable to '00'B, letting the application detect an error.

NOTE 4 A field reserved for future extension should be overwritten with '1'. This sets the Check_Variables that it will once contain to '11'B, and allows future devices not to mistake the '1's for valid data when receiving from older devices.

NOTE 5 The Application is expected to handle the two situations which could arise because of communication problems (all '0') or because data are invalidated (all '1').

EXAMPLE - Figure 11 shows a Process_Variable and its associated Check_Variable in the same Dataset

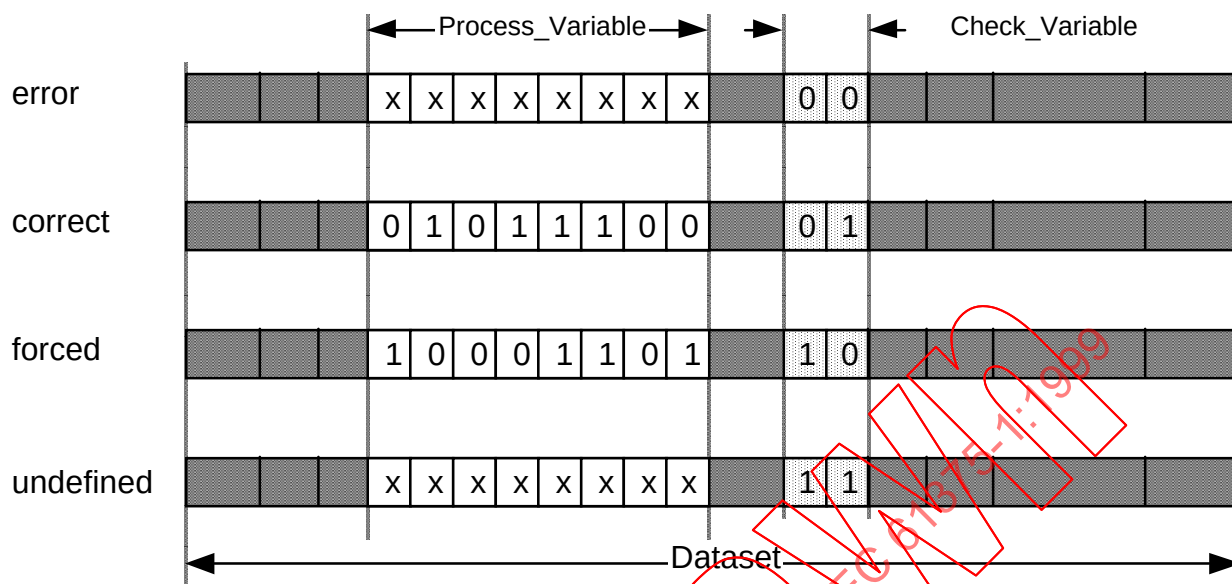


Figure 11 - Check Variable

2.2.3.2.5 Process_Variable identifier (PV_Name)

2.2.3.2.5.1 Variable and Dataset identifier

Within a device, a Process_Variable shall be identified by its Dataset (DS_Name) and its offset in bits within that Dataset (Var_Offset).

When transmitted over a bus, a Process_Variable shall be identified by its Logical_Address and its offset in bits within the transmitted Dataset (Var_Offset).

2.2.3.2.5.2 PV_Name format

Within a device, each Process_Variable shall be identified by a unique identifier, called its PV_Name, consisting of the following elements:

- Traffic_Store_Id;
- Port_Address;
- Var_Offset;
- Var_Size;
- Var_Type;
- Chk_Offset.

NOTE The PV_Name of the same Variable on the same bus could differ in its Traffic_Store_Id from device to device since the Traffic_Store Identifier could vary.

2.2.3.2.5.3 Traffic_Store Identifier

The Traffic_Store_Id shall identify one of the 16 Traffic_Stores within a device.

2.2.3.2.5.4 Port_Address

The Port_Address shall identify one of 4096 ports within the Traffic_Store.

2.2.3.2.5.5 Var_Offset

If a Dataset would contain an unsigned integer, its most significant bit would be at offset 0.

The Var_Offset shall define the offset in bits since the beginning of the Dataset, of the beginning of the field occupied by the value of a Process_Variable.

A Process_Variable shall be located at a Var_Offset which is a multiple of its size.

NOTE Alignment is a concession to compilers which cannot access data structures which spread over a word boundary.

2.2.3.2.5.6 Var_Type and Var_Size

Var_Type and Var_Size shall uniquely identify the format of the Process Variable, 'Var_Type' indicating the type as specified in 2.4 and 'Var_Size' the size.

'Var_Size' and 'Var_Type' shall be encoded as shown in Table 7.

Table 7 - Var_Size and Var_Type encoding in a PV_Name

Var_Size	Var_Type	Data type
0	0	BOOLEAN1
	1	ANTIVALENT2
	2	BCD4 or ENUM4
	3	reserved
	4	BITSET8
	5	UNSIGNED8 or ENUM8
	6	INTEGER8
	7	CHARACTER8 (ARRAY [0..0] OF WORD8)
1	4	BITSET16
	5	UNSIGNED16 or ENUM16
	6	INTEGER16
	8	BIPOLAR2.16 (±200%)
	9	UNIPOLAR2.16 (+400%)
	10	BIPOLAR4.16(±800%)
2	3	REAL32
	4	BITSET32
	5	UNSIGNED32 or ENUM32
	6	INTEGER32
3	2	TIMEDATE48
4	4	BITSET64
	5	UNSIGNED64
	6	INTEGER64
n-1	7	ARRAY OF WORD8 (odd number of octets)
	15	ARRAY OF WORD8 (even number of octets)
	13	ARRAY OF UNSIGNED16 (n = array size in WORD16)
	14	ARRAY OF INTEGER16
	11	ARRAY OF UNSIGNED32 (n = array size in WORD16)
	12	ARRAY OF INTEGER32

'Var_Size' is interpreted differently in structured and in primitive types:

- in primitive types, 'Var_Size' indicates the number of 16-bit words used;
- in structured types, 'Var_Size' indicates the number of 16-bit words minus 1.

NOTE 1 In primitive types, Var_Size = 0 is less than one WORD16, Var_Size = 1 is one WORD16.

NOTE 2 In structured types, Var_Size = 0 is one WORD16.

NOTE 3 The factor 'n' is the number of WORD16. The maximum size of a variable is therefore 128 octets (64 x 16 = 1024 bits), Var_Size being '3F'H.

NOTE 4 Codes not mentioned in Table 7 are reserved.

NOTE 5 Although the triple {Traffic_Store, Port_Address, Var_Offset} uniquely identifies a Process_Variable, the PV_Name includes the type/size information to convert rapidly between network and application data types.

2.2.3.2.5.7 Chk_Offset

The Chk_Offset shall define the position of the Check_Variable associated with the Process_Variable with respect to the beginning of the Dataset.

A Chk_Offset corresponding to the rightmost position in the Dataset ('0FFF'H) shall be used when a Process_Variable has no associated Check_Variable.

2.2.3.3 Application_Variables_Interface primitives

2.2.3.3.1 General

The Application_Variables_Interface (AVI) primitives are divided into three groups:

- individual access;
- Set access;
- Cluster access.

2.2.3.3.2 Type 'AP_RESULT'

Definition	A procedure of the AVI which returns a value shall encode it as follows:
Syntax	<pre> typedef enum { AP_OK = 0, /* correct termination AP_PRT_PASSIVE = 1, /* warning: Dataset not active AP_ERROR = 2, /* general error AP_CONFIG = 3, /* configuration error occurred AP_MEMORY = 4, /* not enough memory to complete AP_UNKNOWN_TS = 5, /* unknown Traffic_Store AP_RANGE = 6, /* Memory_Address error AP_DATA_TYPE = 7 /* unsupported data type } AP_RESULT; </pre>

NOTE The encoding of these constants is identical to the LPI constants with similar name.

2.2.3.3.3 Configuration Parameters

AP_TS_ID_MAX	0..15	maximum number of Traffic_Stores supported in an implementation = AP_TS_ID_MAX + 1

2.2.3.3.4 Variable initialisation

The initialisation of the Process_Variables is performed by the Dataset initialisation mechanism for all Datasets, which is application-dependent.

NOTE The link layer is expected to initialise by default all Datasets to '0'.

2.2.3.3.5 Individual access primitives

The Application Layer for Variables (AVI) shall provide for individual Variable access the following primitives, illustrated in Figure 12 and specified in the following subclauses:

- ap_put_variable,
- ap_get_variable,
- ap_force_variable,
- ap_unforce_variable,
- ap_unforce_all.

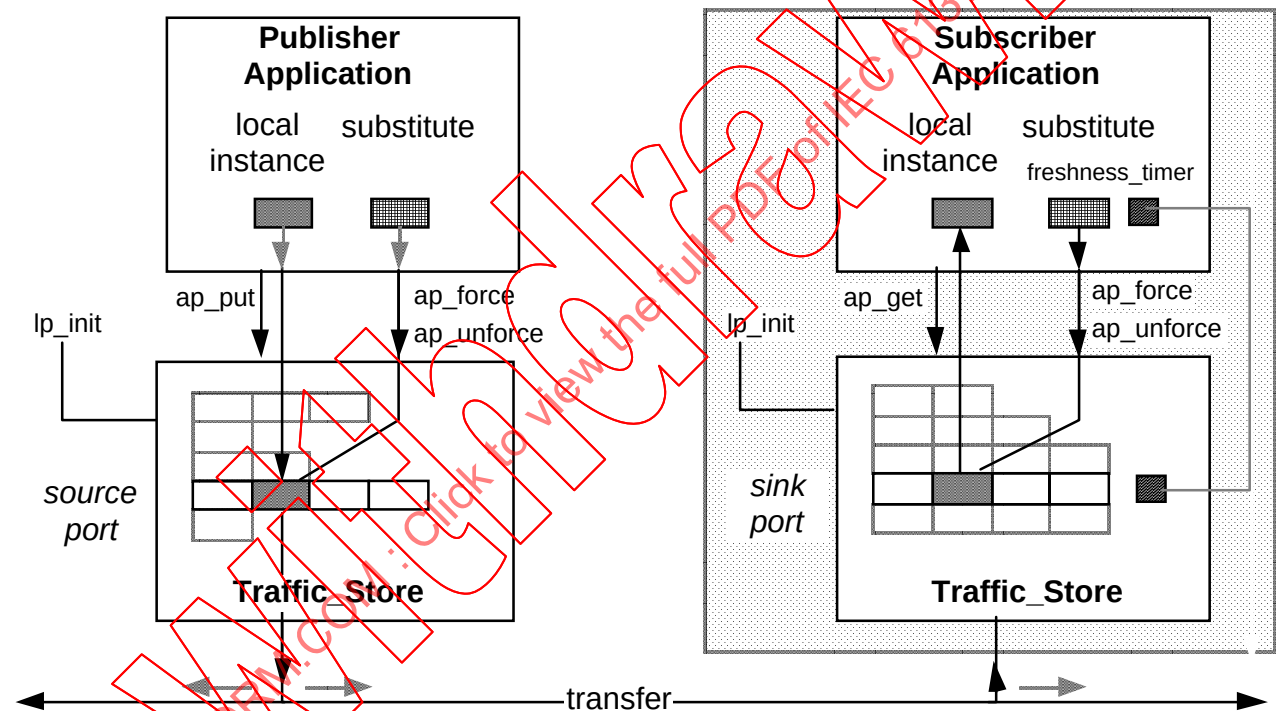


Figure 12 - Individual access

2.2.3.3.5.1 Type PV_NAME

For individual access, Var_Offset and Chk_Offset shall consist each of two fields, Var_Octet_Offset and Var_Bit_Number.

Var_Octet_Offset is the octet offset with respect to the beginning of the dataset, the first transmitted or stored octet being octet 0.

In variables which consist of several octets, Var_Bit_Number is always 0. In Variables smaller than one octet, Var_Bit_Number is the number of bits the variable shall be shifted right so that the variable is right-justified in an octet. Var_Bit_Number is not identical with the bit offset within the octet.

Definition	Type of an individual Process_Variable
Syntax	<pre> typedef struct /* big-endian representation */ { unsigned traffic_store_id :4, /* DS_NAME first part */ unsigned port_address :12, /* DS_NAME second part */ unsigned var_size :6, /* as given in Table 2.2 */ unsigned var_octet_offset :7, /* first octet has offset 0 */ unsigned var_bit_number :3, /* counted from the right */ unsigned var_type :6, /* as given in Table 2.2 */ unsigned chk_octet_offset :7, /* first octet has offset 0 */ unsigned chk_bit_number :3, /* counted from the right */ } PV_NAME; </pre>

The PV_Name can be conveniently encoded as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Traffic_store_id				port_address											
var_size						var_octet_offset						var_bit_number			
var_type						chk_octet_offset						chk_bit_number			

NOTE The definition of Var_Bit_Number was introduced to speed up individual access and especially avoid to add or subtract eight to take advantage of the shift instructions in the processors. This decomposition is only possible if all data types are aligned, for instance, an ANTIVALENT2 may not be at an odd offset.

EXAMPLE:

The following memory dump represents a PV_Name:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	1	1	0	1	1	1	0	1	0
0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0

This PV_Name identifies a Process_Variable which:

- is located in Traffic_Store 3, at port_address 'BA'H (= 442), at bit offset '0F8'H (31 x 8 = 248) bits;
- it is of type INTEGER8 (var_size = 0 x WORD16, var_type = 6);
- its associated 2-bit Check_Variable is located at octet offset 0, bit number 4, i.e. it is located in the 3rd and 4th bits of the dataset (bit offset 2 and 3). If this number would be odd, there would be no associated check variable.

2.2.3.3.5.2 Procedure ap_put_variable

Definition	Copies an individual Process_Variable and its associated Check_Variable from the Application Memory_Address space to a Traffic_Store.	
Syntax	<pre> AP_RESULT PV_NAME* void * void * </pre>	<pre> ap_put_variable (ts_variable, p_value, p_check); </pre>
Input	ts_variable	PV_Name of the Process_Variable
	p_value	pointer to a memory location of the Application where the published value is copied from.
	p_check	pointer to a memory location of the Application where the associated Check_Variable is copied from.
Return		any AP_RESULT
Usage	1 - If the Process_Variable has been forced, ap_put_variable has no effect. 2 - The former value of the Process_Variable is overwritten. 3 - Other data of the same Dataset are not affected, but consistency with them is not guaranteed.	

2.2.3.3.5.3 Procedure ap_get_variable

Definition	Copies a Process_Variable and its associated Freshness_Timer and Check_Variable from a Traffic_Store to the Application.	
Syntax	<pre> AP_RESULT PV_NAME* void * void * void * </pre>	<pre> ap_get_variable (ts_variable, p_value, p_check, p_fresh); </pre>
Input	ts_variable	PV_Name of the Process_Variable
	p_value	pointer to a memory location of the Application where the received value is put.
	p_check	pointer to a memory location of the Application where the associated Check_Variable is put.
	p_fresh	pointer to a memory location of the Application where the associated Freshness_Timer is put.
Return		any AP_RESULT
Usage	1 - This primitive may be used with a source or a sink port, to allow Subscribers on the same device as the Publisher. 2 - If the Process_Variable has been forced, the forced value is retrieved.	

2.2.3.3.5.4 Procedure ap_force_variable

Definition	Forces an individual Process_Variable in a port to a specified value; puts the associated Check_Variable to the value '10'B.	
Syntax	<pre> AP_RESULT ap_force_variable (PV_NAME * ts_variable, void * p_value); </pre>	
Input	ts_variable	PV_Name of the Process_Variable
	p_value	pointer to a memory location of the Application where the forced value is copied from.
Return		any AP_RESULT
Usage	The substituted value is expected to have a type compatible with that of type PV_NAME.	

2.2.3.3.5.5 Procedure ap_unforce_variable

Definition	Terminates the forcing of a Variable and restores normal bus access to it; does not modify the corresponding Check_Variable.	
Syntax	<pre> AP_RESULT ap_unforce_variable (PV_NAME * ts_variable); </pre>	
Input	ts_variable	PV_Name of the Process_Variable
Return		any AP_RESULT

2.2.3.3.5.6 Procedure ap_unforce_all

Definition	Terminates the substitution of all Variables of a Traffic_Store, does not modify the corresponding Check_Variables.	
Syntax	<pre> AP_RESULT ap_unforce_all (ENUM8 ts_id); </pre>	
Input	ts_id	Traffic_Store_Id (0..15)
Return		any AP_RESULT

2.2.3.3.6 Set access procedures**2.2.3.3.6.1 Set access mode**

A Set is a group of Variables (Process_Variables and Check_Variables) belonging to the same Dataset and treated as a whole to preserve consistency and freshness information.

The Application Layer for Variables (AVI) shall provide for Set access the following primitives, illustrated in Figure 13 and specified in the following subclauses:

- a) ap_put_set,
- b) ap_get_set.

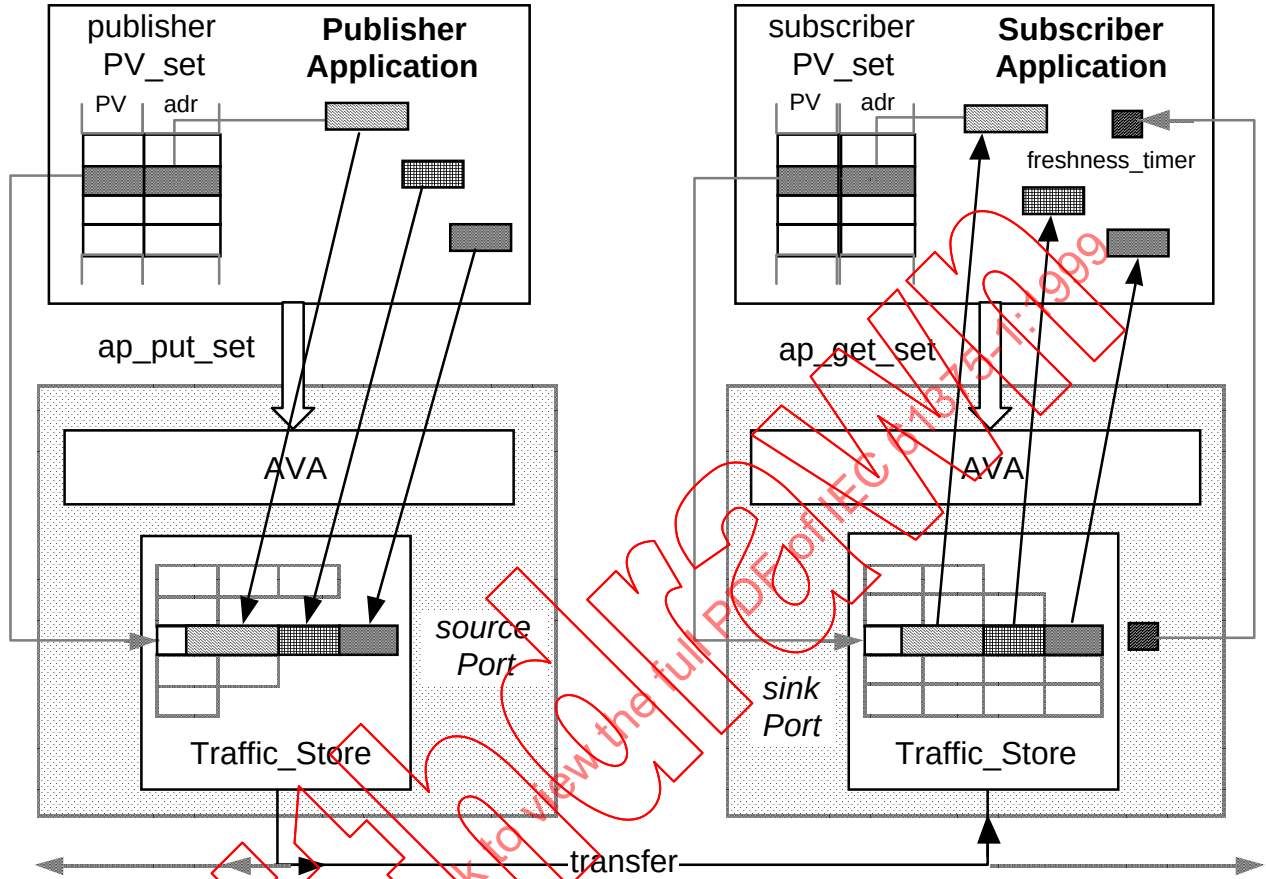


Figure 13 - Set access

NOTE Set access is consistent, i.e. all variables are copied or none in one indivisible operation.

2.2.3.3.6.2 Type PV_SET

Definition	A PV_Set identifies a set of Variables belonging to the same Dataset, including for each Variable the Memory_Address where it should be copied to (or from), and including for the whole Dataset the Freshness_Timer.	
Syntax	typedef struct	PV_LIST { void* p_variable; UNSIGNED8 derived_type; UNSIGNED8 array_count; UNSIGNED8 octet_offset; UNSIGNED8 bit_number; };
	typedef struct PV_LIST* UNSIGNED16 UNSIGNED16 * DS_NAME	PV_SET { p_pv_list; c_pv_list; p_freshtime; dataset; };
Elements	p_variable	Memory_Address of the Variable
	derived_type	generalised data type, derived from Var_Type and Var_Size, implementation-dependent.
	array_count	number of elements in the array.
	octet_offset	offset in number of octets of a Variable
	bit_number	bit number of a Process_Variable smaller than one octet or Check_Variable (see PV_Name definition)
	p_pv_list	pointer to PV_List
	c_pv_list	number of Variables in the PV_List
	p_freshtime	Memory_Address of the Freshness_Timer. (not used for ap_put_set)
	dataset	DS_Name (holds for the whole Set)
Usage	1 - Process_Variables and Check_Variables are treated identically, since all Variables of a PV_Set are consistent. Therefore, there is no distinction between Var_Offset and Chk_Offset. 2 - The Var_Offset (or Chk_Offset) is divided into an octet offset and a bit offset for faster processing. For the same reason, type and size each occupy one octet instead of the six bits used in the type PV_NAME. 3 - The same format is used for the Subscriber Set and for the Publisher Set, although the Freshness Counter is not used in a Publisher Set. 4 - To increase access efficiency, the PV_Set may also contain direct references to internal data structures of a Traffic_Store.	

NOTE For efficiency reasons, the PV_SET does not include the full PV_NAME of each Variable. In particular, the Check_Variable appears as a normal ANTIVALENT2, since the same Check_Variable can protect several Variables.

2.2.3.3.6.3 Procedure ap_put_set

Definition	Copies a list of Variables belonging to the same Set from the Application Memory_Address space to the port, in an indivisible operation.	
Syntax	<pre> AP_RESULT ap_put_set (PV_SET * pv_set); </pre>	
Input	pv_set	pointer to a PV_List
Return		any AP_RESULT

2.2.3.3.6.4 Procedure ap_get_set

Definition	Copies a list of Variables belonging to the same Set from the port to the Application Memory_Address space, in an indivisible operation.	
Syntax	<pre> AP_RESULT ap_get_set (PV_SET * pv_set), </pre>	
Input	pv_set	pointer to a PV_List
Return		any AP_RESULT

2.2.3.3.7 Cluster access procedures

2.2.3.3.7.1 Cluster access mode

Clusters are groups of Variables scattered over several Datasets and over several Traffic_Stores.

The Application Layer for Variables (AVI) shall provide for Cluster access the following primitives, illustrated in Figure 14 and specified in the following subclauses:

- ap_put_cluster,
- ap_get_cluster.

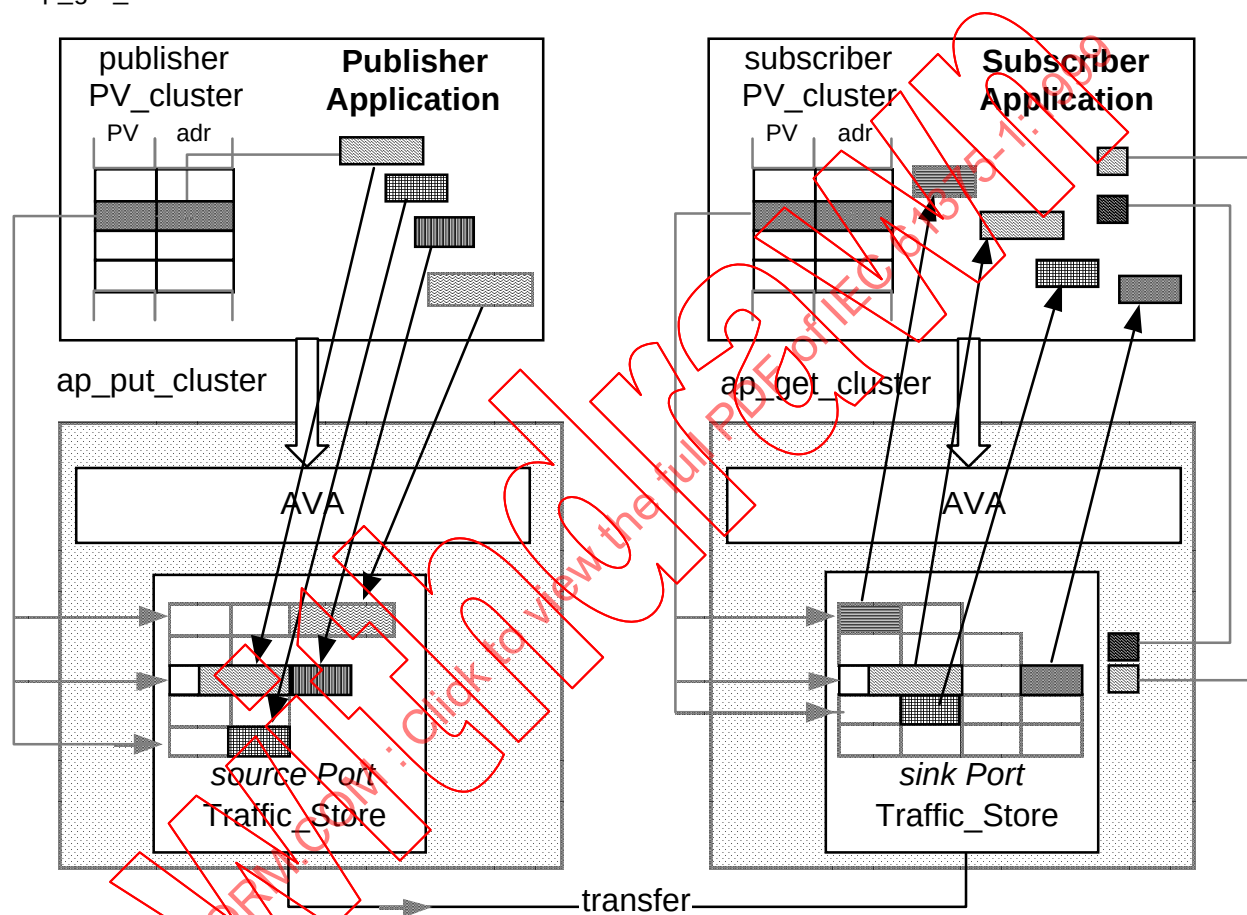


Figure 14 - Cluster access

NOTE 1 Cluster access does not guarantee consistency of the whole Cluster, but only that of the individual PV_Sets in the Cluster.

NOTE 2 No consistency is guaranteed between Variables of the same Dataset which appear in different PV_Sets.

2.2.3.3.7.2 Type PV_CLUSTER

A PV_Cluster identifies a group of PV_Sets, ordered by Traffic_Stores.

Definition	Type of a PV_Cluster	
Syntax	<pre>typedef struct PV_CLUSTER { UNSIGNED8 ts_id; UNSIGNED8 c_pv_set; struct PV_SET * p_pv_set [c_pv_set] }</pre>	
Elements	ts_id	Traffic_Store_Id (0..15) of one Traffic_Store
	c_pv_set	number of PV_Sets in the Cluster
	p_pv_set	ARRAY [0..c_pv_set-1] OF pointers to PV_SET
Usage	1 - There is a Cluster List for each Traffic_Store. 2 - The same format is used for the Subscriber Cluster List and for the Publisher Cluster List, although the Freshness Counter is not used in a Publisher Cluster List. 3 - To increase access efficiency, the PV_Cluster may also contain direct references to internal data structures of a Traffic_Store.	

2.2.3.3.7.3 Procedure ap_put_cluster

Definition	Copies a Cluster of Variables from the Application to the Traffic_Store. Variables belonging to the same PV_Set are copied consistently.	
Syntax	<pre>AP_RESULT ap_put_cluster (PV_CLUSTER* pv_cluster);</pre>	
Input	pv_cluster	pointer to a Publisher Cluster List
Return		any AP_RESULT

2.2.3.3.7.4 Procedure ap_get_cluster

Definition	Copies a Cluster of Process_Variables from the Traffic_Store(s) to the local Subscriber instances. Variables belonging to the same PV_Set are copied consistently.	
Syntax	<pre>AP_RESULT ap_get_cluster (PV_CLUSTER* pv_cluster);</pre>	
Input	pv_cluster	pointer to a Subscriber Cluster List
Return		any AP_RESULT

2.3 Messages Services and Protocols

2.3.1 General

For an introduction to Messages, see subclause A.2.3.

The Messages services and protocols are separated in a lower interface and a higher interface:

- the lower, link layer interface, which specifies the services expected from the bus; and
- the higher, application layer services are offered to the application.

2.3.2 Reference station

A TCN station providing the Messages Services shall include the following elements:

- at least one bus connection, accessible through its LPI, and which implements a link layer Process;
- a protocol machine, called the Messenger, implementing the network, transport, session, presentation and application layer;
- one or several Application Processes, one of them being the Network Management Agent.

A Manager station shall provide a Manager Application Process.

NOTE The minimum set of services an Agent is expected to provide is specified in Clause 5 (Train_Network_Management).

2.3.2.1 Terminal station

Stations connected to one bus only, or terminal stations, shall have only one link layer.

EXAMPLE - The structure of a reference terminal station is shown in Figure 15.

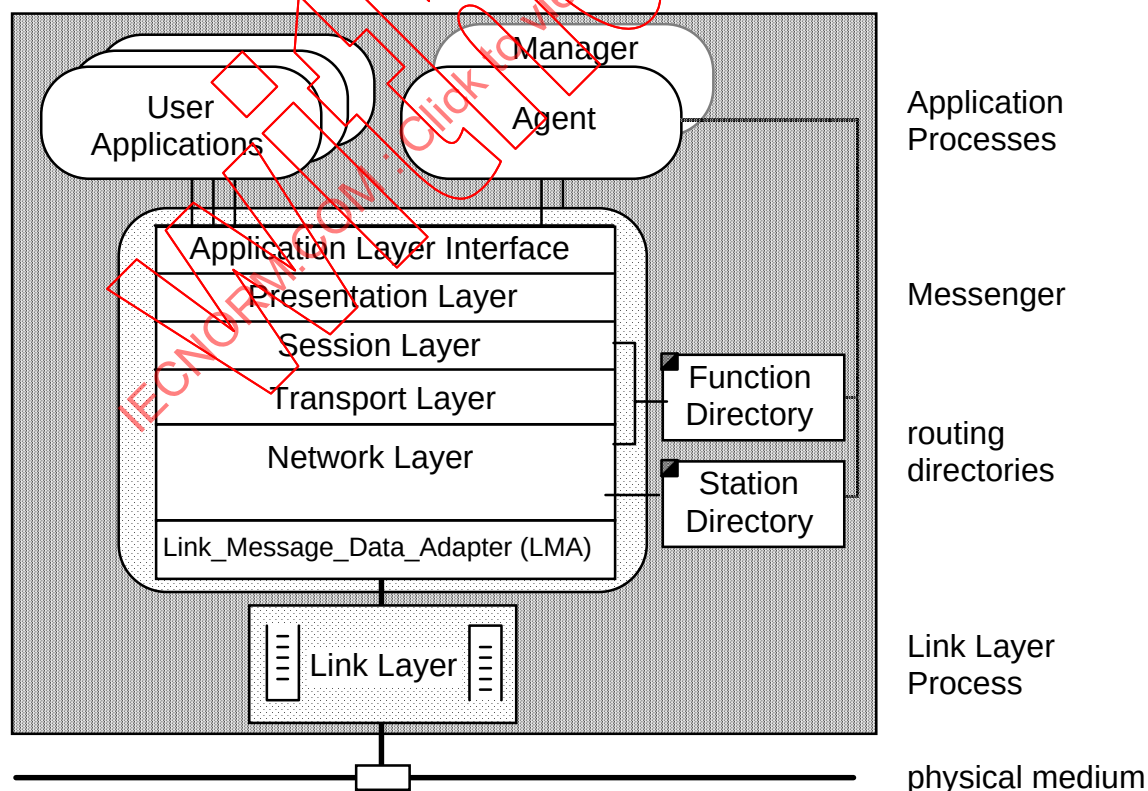


Figure 15 - Terminal station

NOTE The network layer of terminal stations only accesses the own transport layer and the link layer.

2.3.2.2 Router station

Station attached to several busses, or Router station, shall have one link layer for each bus.

EXAMPLE - Figure 16 shows a Router station attached to the link layer of an MVB and of a WTB.

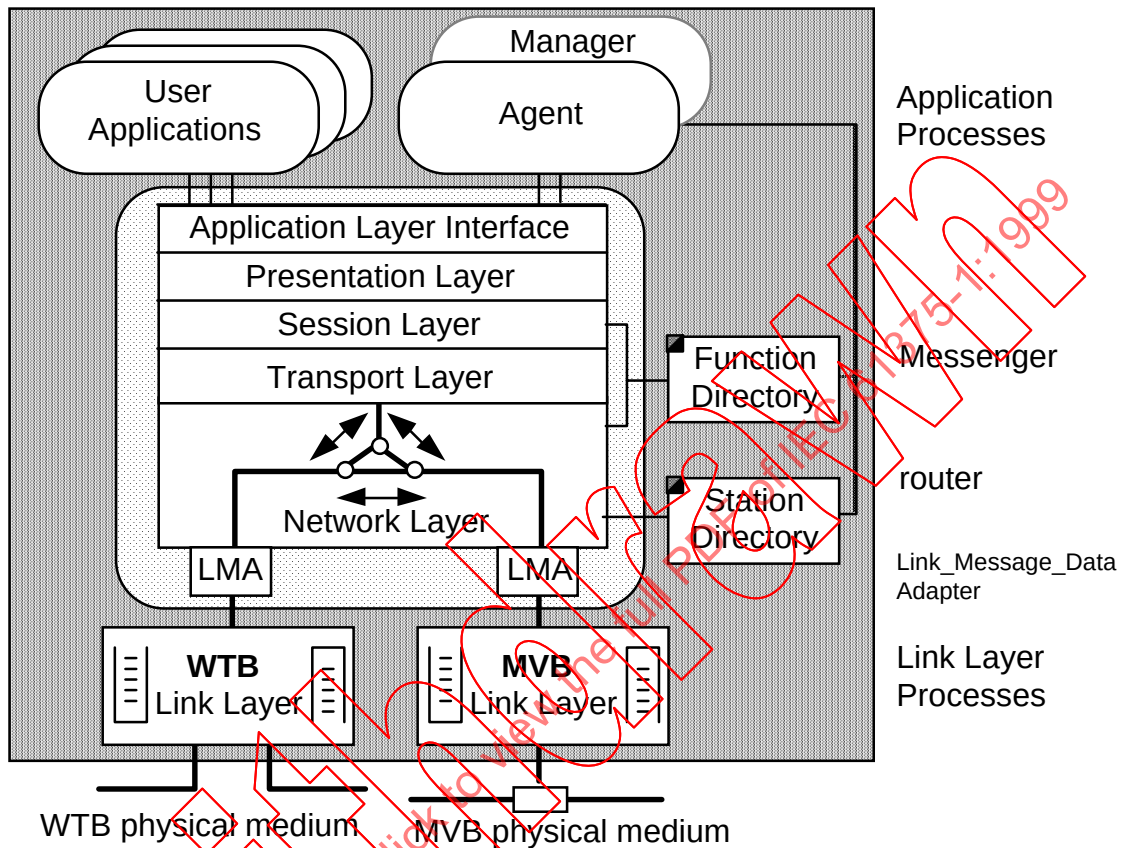


Figure 16 - Router station between WTB and MVB

NOTE The network layer of a terminal station is able to route packets from bus to bus.

2.3.2.3 Station identifier

A station shall be identified by a Station_Id.

Since there may be only one Agent and one Messenger per station, the Station_Id also identifies the Agent.

NOTE The station identifier is not necessarily identical to a Device_Address. A router station has one Device_Address per bus to which it is attached, but only one Station_Id. The station identifier can be modified by Network Management, while the Device_Address is often hard-wired.

2.3.2.4 Bus identifier

A station shall identify each link layer, i.e. each bus (vehicle bus or train bus), to which it is connected by a Bus_Id.

The maximum number of busses connected to a station shall be 16.

NOTE Bus_Id does not imply which type of bus is attached, but it can be helpful to give always Bus_Id = 1 to the train bus.

2.3.2.5 Link address

A station shall identify each device which it can access over one of its busses by a Link_Address.

The Link_Address shall consist of the concatenation of the Bus_Id and of the Device_Address of the device.

NOTE The size of the Device_Address depends on the bus.

2.3.3 Message packets handling

Packet handling within a communication protocol stack is an implementation issue. The interface procedures in the Messages Services specify a message packet through a pointer. It is left to the implementation to use this pointer to pass the packet contents by reference or by copying them. The packet structure itself is not prescribed.

To ease portability and explain some interface procedures, certain packet handling procedures are defined in this subclause.

This interface does not need to be exposed and it is not subject to Conformance Test.

2.3.3.1 Packet pools

To ensure a consistent access within a station, all packets used by the transport layer, network layer and link layer have the same format.

To pass packets by reference rather than copy them, a dynamic memory management of packets is needed, in form of packet pools. Initially, a packet pool is created with a number of void packets. A user may request packets from the pool and returns them to the pool after use. The net flow of packets to and from a pool should be zero in the long run.

There exist several packet pools, typically a pair for each link layer. The pool to which a packet belongs is its Owner.

2.3.3.2 Type 'MD_PACKET'

Packets are identified by a packet descriptor, which point to the data of the packet to be sent as well as some more fields which are used for packet management.

A packet is referenced by a packet pointer, which is unique to that packet, and which points to the packet descriptor, which is of type MD_PACKET.

Definition	This type defines the packet format.
Syntax	<code>typedef void* MD_PACKET;</code>
Usage	The format of MD_PACKET is not prescribed.

EXAMPLE - An example of MD_PACKET is shown in Figure 17.

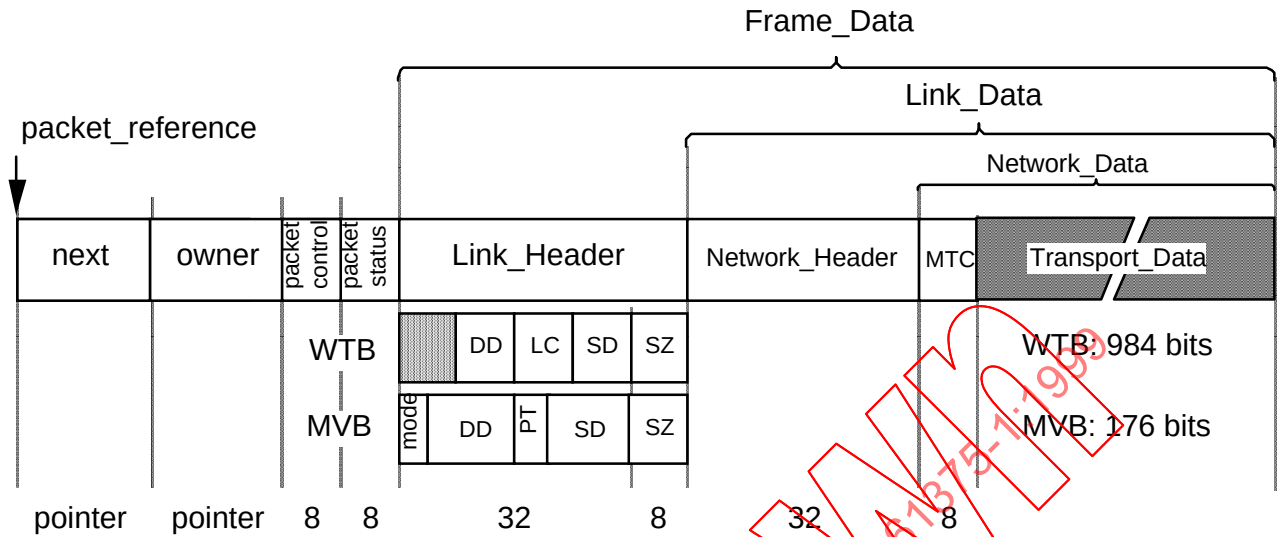


Figure 17 - Packet format

The example packet of Figure 17 begins with the following fields:

- 'next': The first field in the packet is reserved for a pointer to another packet. This allows linking of packets into lists or queues.
- 'owner': The second field identifies the owner (pool) of the packet. This field is used to dispose of a used packet.
- 'packet control' contains management information which the link layer may read, but not modify.
- 'packet status' is modified by the link layer and can be read by the other layers.

The rest of the packet contains the frame actually sent or received over the bus:

- The Link_Header has a different format depending on the bus (MVB, WTB or other). It contains the source Device_Address and the destination Device_Address, among other bus-specific control information.
- The Link_Header is only of interest to the link layer, it is not analysed by the network layer, which receives this information over parameters.
- The 'size' field applies to the Link_Data, excluding the field size itself.

The status of a packet may be:

- MD_PENDING this packet is marked for sending;
- MD_FLUSHED this packet has been flushed from a queue;
- MD_SENT this packet has been sent and can be recycled.

2.3.3.3 Procedure Type 'MD_GET_PACKET'

Definition	gets a new packet from a pool, sets the owner field of the packet to the value 'pool'.	
Syntax	<pre>typedef void (* MD_GET_PACKET) (void * * MD_PACKET * * pool, packet);</pre>	
Input	pool	Identifies the pool which the packet is taken from
Output	packet	Pointer to a data structure where the packet is stored.
Usage	This procedure type is compatible with the LM_GET_PACK procedure type, which can be directly called.	

2.3.3.4 Procedure Type 'MD_PUT_PACKET'

Definition	returns a single packet not anymore used to the pool specified by the owner field of the packet.	
Syntax	<pre>typedef void (* MD_PUT_PACKET) (MD_PACKET * packet);</pre>	
Input	packet	Pointer to a data structure where the packet can be found. The owner pool is marked in the packet.
Usage	This procedure type is compatible with the LM_SEND_CONFIRM procedure type, which can be directly called.	

2.3.4 Message Link layer

2.3.4.1 Purpose

The Message Services are provided on different busses: WTB, MVB or other, including parallel busses, serial links, memory mailboxes or sensor busses.

To this effect, the link layer of any of these busses is expected to provide a set of basic services, which are specified in this subclause.

Generally, the link layer of one device co-operates with the link layer of another device to exchange packets between a source device and a destination device located on the same bus as shown in Figure 18.

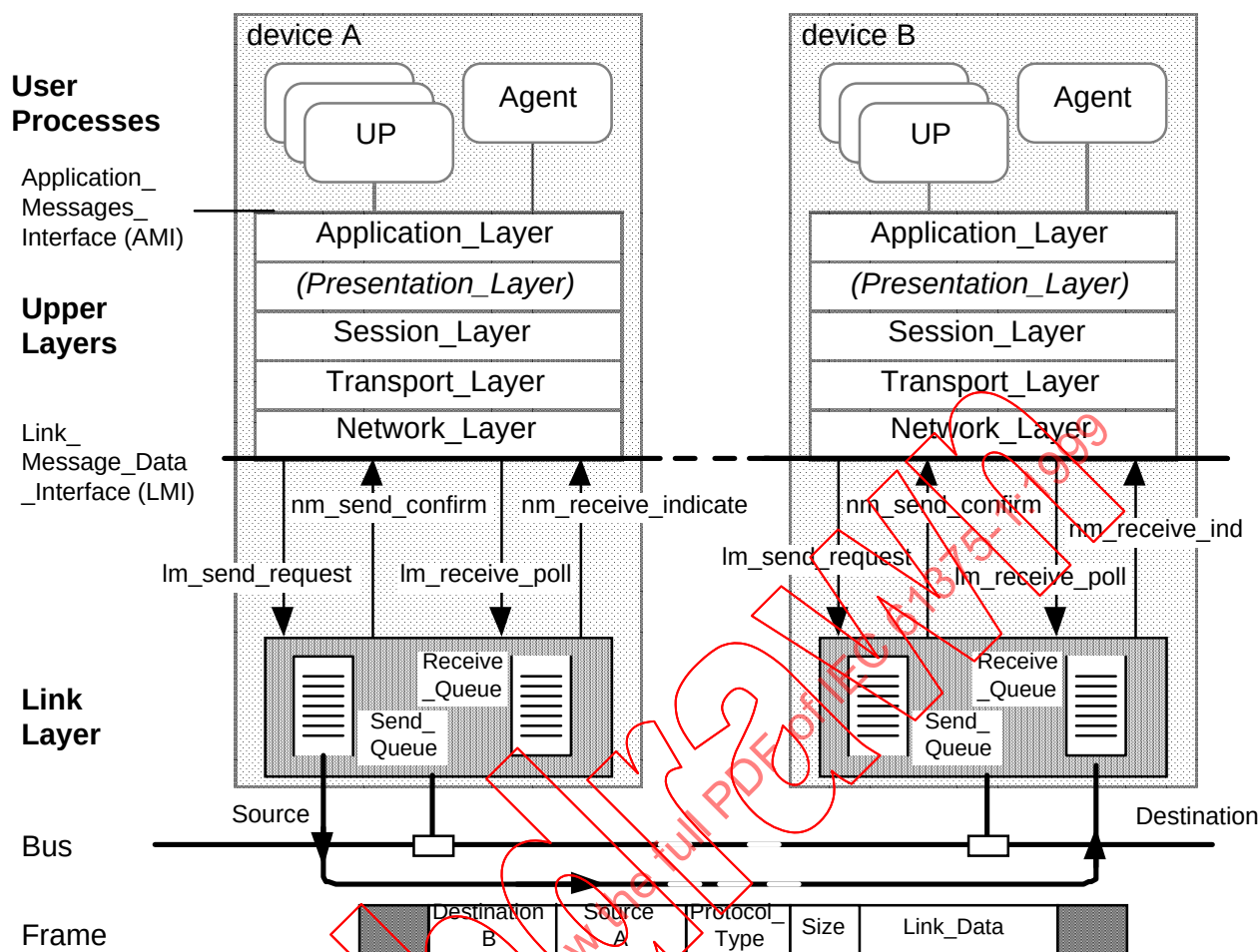


Figure 18 - Link layer data transmission

The link layer process executes the actual transmission over the bus. It can run independently from the Applications and from the Messenger.

The implementation of the link layer is not specified. Therefore, the services of the link layer are specified in a general form which is expected to be found in every bus attached to the Messenger.

2.3.4.2 Link layer structure

A link layer shall provide a pair of queues, a Send_Queue in which the network layer puts packets to be sent and a Receive_Queue in which the network layer retrieves packets received from the bus.

A (router) station may have several link layers, one for each attached bus, as shown in Figure 19.

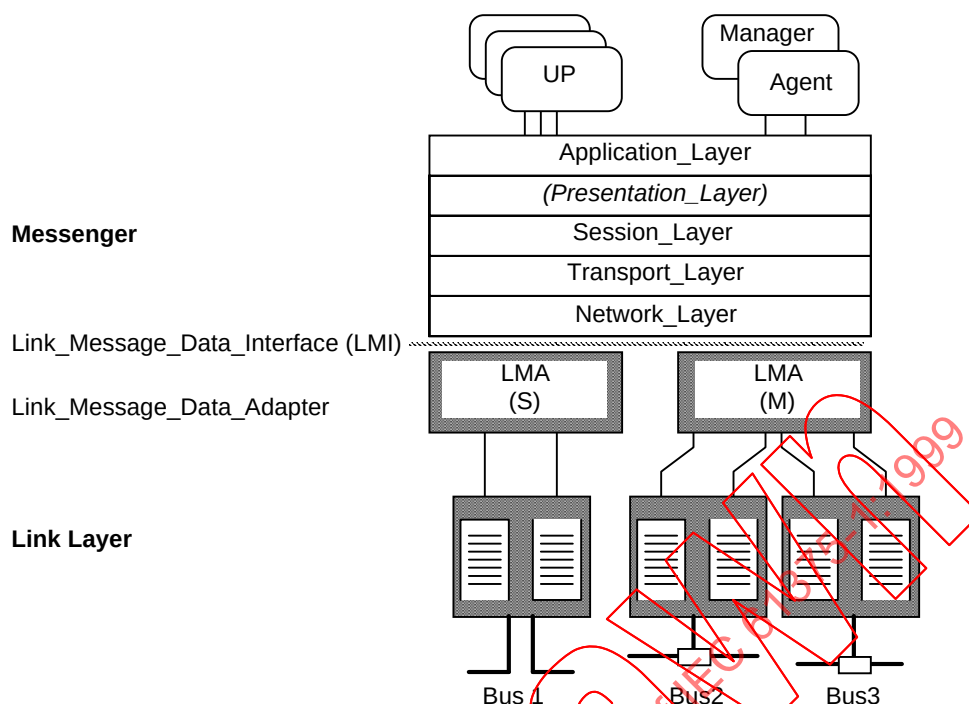


Figure 19 - Link_Message_Data_Interface (LMI)

The differences in the implementation of the link layers are hidden in the Link_Message_Data_Adapter (LMA).

NOTE An LMA may support one single bus only (Type S), or several busses (Type M).

2.3.4.3 Link layer characteristics

2.3.4.3.1 Device address

Each device shall be uniquely identified on a bus by its Device_Address.

Device_Address 0 shall identify the local link layer and shall not be allocated to a specific device.

The highest Device_Address (e.g. '11111111'B for an 8-bit Device_Address) shall indicate a broadcast to all devices on the bus and shall not be allocated to a specific device.

The link layer shall include its own Device_Address as Source_Device in all packets it sends, but it shall not use the bus-specific broadcast address as Source_Device.

The link layer shall include the Device_Address of the remote device or the bus-specific broadcast address as Destination_Device in all packets it sends.

NOTE 1 The format of the Device_Address depends on the bus (WTB, MVB or other).

NOTE 2 A router device has one Device_Address for each bus attached to it.

2.3.4.3.2 Protocol type

The link layer for messages shall have the form of a pair of queues (consumable buffers).

If other protocols are supported in the same device, a Protocol_Type (PT) in each frame shall select the Send_Queue and the Receive_Queue used for the Real-Time Protocols.

NOTE Protocol_Type plays a role similar to a Link Service Access Point in ISO/IEC 7498.

2.3.4.3.3 Priorities

Priorities are not distinguished at the link layer.

NOTE The MVB allows two event priorities, but only one is used for Message_Data.

2.3.4.3.4 Flushing

The link layer of a Node shall include provision to flush all packets.

NOTE Flushing is necessary in case a Node receives a new Topography.

2.3.4.3.5 Packet lifetime

The link layer shall include provision to limit the lifetime of the packets in its Receive_Queue or Send_Queue to a time less than PACK_LIFE_TIME.

PACK_LIFE_TIME shall be 5,0 s.

NOTE Limitation of life-time may be achieved by a queue time-out (queue flush) or by a individual invalidation of the packet in the queue.

2.3.4.3.6 Link layer protocol

The link layer protocol shall be connectionless, i.e. operate only with datagrams.

The link layer shall not automatically repeat lost frames.

The link layer shall register transmission errors for network management.

2.3.4.4 Link layer of the vehicle bus

The vehicle bus may be implemented by different busses, in particular by the MVB (see Clause 3), which serves as a reference.

The Device_Address of a vehicle bus device shall not change during regular operation.

If the MVB is used as a vehicle bus, the following additional specifications apply:

- the 12-bit Device_Address shall identify destination and source devices;
- the Mode field shall specify a single cast transmission as '0001'B or a broadcast transmission as '1111'B. In the second case, the Device_Address is 'don't care';
- if the highest Device_Address ('111111111111'B) is specified, the broadcast transmission mode shall be selected;
- The 4-bit Protocol_Type '1000'B shall identify the TCN Real-Time-Protocols.

EXAMPLE - The format of a Message_Data frame over the MVB is given by Figure 20.

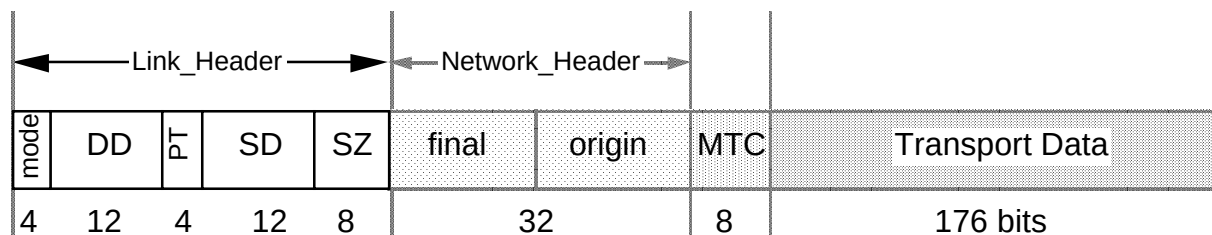


Figure 20 - Example of MVB Message_Data frame

2.3.4.5 Link layer of the train bus

The train bus may be implemented by different busses, in particular by the WTB (Clause 4) which serves as reference. One distinguishes train busses with variable composition (such as the WTB) and train busses with fixed composition (such as the MVB).

2.3.4.5.1 Train bus with variable composition

In a train bus with variable composition, in which the Node addresses may change dynamically, the link layer shall notify its network layer of a change and supply to it the Topo_Counter, a 6-bit counter which is incremented (modulo 64) each time composition changes.

NOTE 1 The train bus may supply additional information, such as the Topography, through link layer Management Services, specified in the WTB (Clause 4). Although the Messages services consider only the Topo_Counter, the Application needs the Topography to select the correct Node addresses in function of the type of vehicle. The mapping of vehicle to Node addresses is an application issue.

NOTE 2 Since it is possible that an Application is not aware that a new Topography was distributed since it read the old one, the Topo_Counter is used to qualify the Topography information.

In case the WTB is used as a train bus, the following additional specifications hold.

The WTB Nodes addresses Nodes through the 8-bit Node_Address assigned by the inauguration.

The WTB uses as broadcast address Destination_Device = '1111111'B.

The Protocol_Type of the link layer for the Real-Time Protocols is identified by Link_Control = 'x0xx111'B (Message_Data).

EXAMPLE - Figure 21 shows the frame format of the WTB.

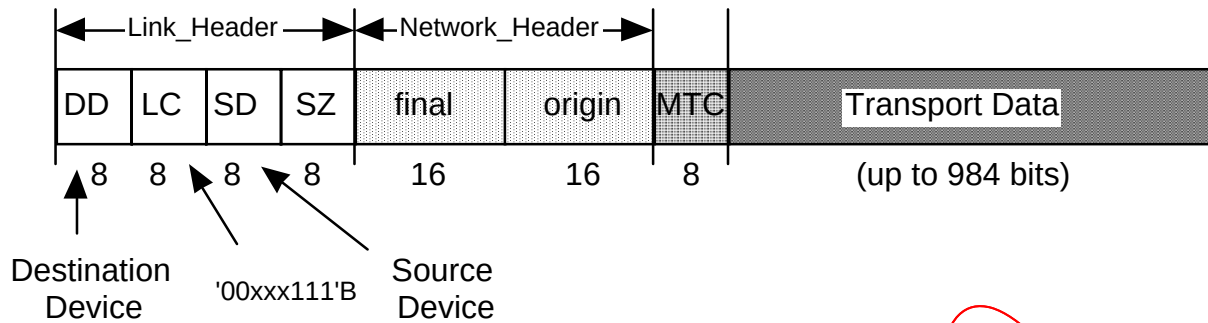


Figure 21 - Example of WTB Message_Data frame

2.3.4.5.2 Train bus with fixed composition

In applications which do not require dynamic allocation of addresses, another train bus than the WTB can be used, for instance the MVB.

To this effect, the Node_Addresses are mapped to the Device_Addresses of the train bus through a node directory.

Since configuration is fixed, the Topo_Counter has no role, and this field may be used instead as an address extension to the Node_Address.

2.3.4.6 Link layer interface for Message Data (LMI)

The Link_Message_Data_Interface (LMI) defines the services which a link layer offers to the network layer.

This interface is not covered by Conformance Testing. It is specified in the following subclauses to ease porting.

2.3.4.6.1 LMI primitives

The Link_Message_Data_Interface shall provide the primitives illustrated in Figure 22, which are listed in Table 8 and specified in the following subclauses.

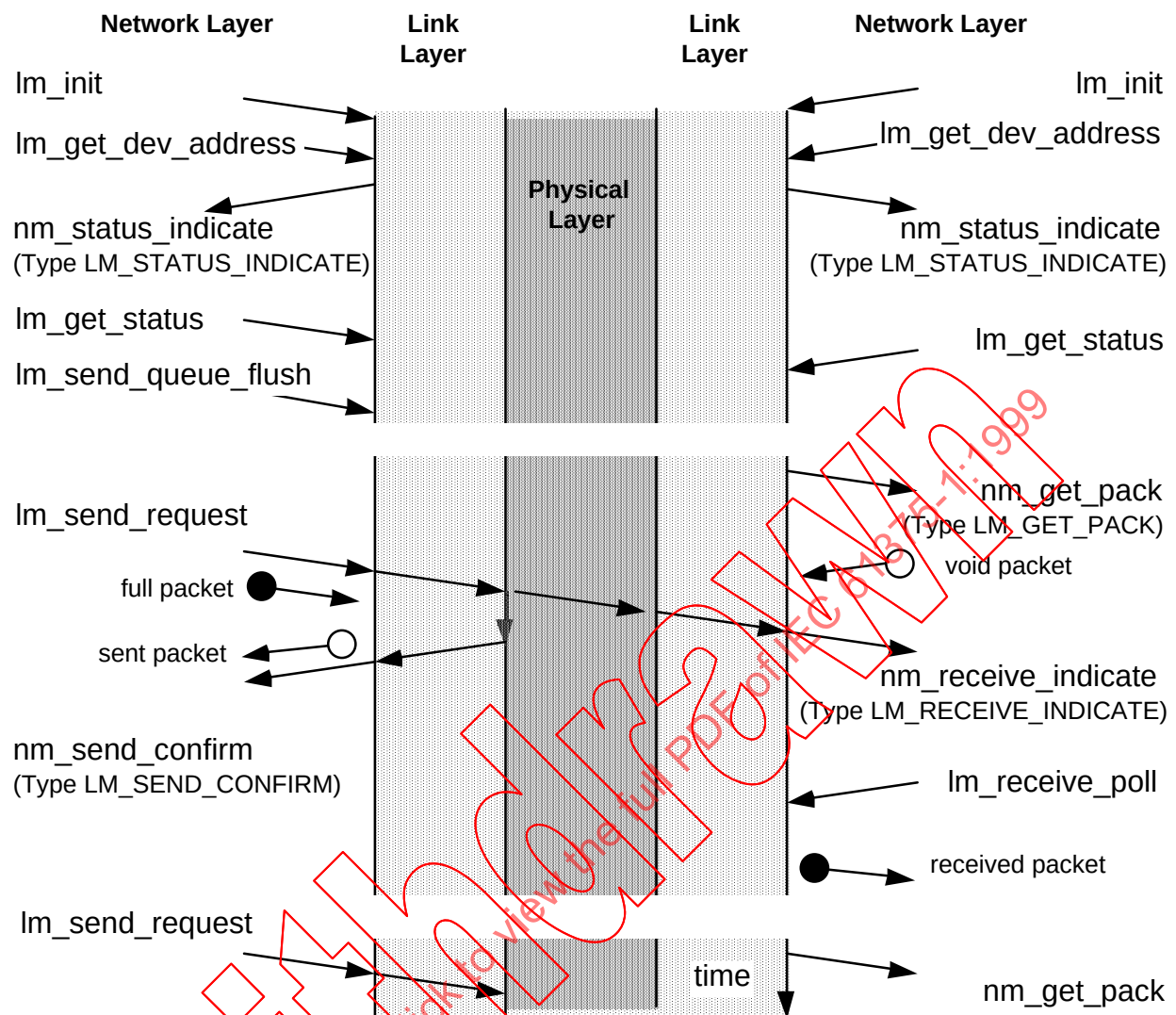


Figure 22 - LMI primitives

NOTE The procedures of the LMI are prefixed by `lm_`, the types of the link layer are prefixed by `LM_`, and the primitives of the network layer by prefixed by `nm_`.

The link layer calls indication procedures of the network layer (prefixed by `nm_`) which were previously subscribed and which have a type defined by the link layer (prefixed by `LM_`).

Table 8 - LMI primitives

Name	Meaning
MD_RESULT	Result of a procedure
lm_send_request	Request to send a packet
LM_SEND_CONFIRM	Indication that a packet was sent
LM_GET_PACK	get a void packet
LM_RECEIVE_INDICATE	Indicate a packet was received
lm_receive_poll,	polls for received packets
LM_STATUS_INDICATE,	Indicates change of status
lm_get_status,	Retrieves link layer status
lm_send_queue_flush,	Flushes the Send_Queue
lm_init	Initialises the link layer
lm_get_dev_address.	Reads the Device_Address

2.3.4.6.2 Type 'MD_RESULT'

Definition	A procedure of the LMI which returns a value shall encode it as follows:
Syntax	<pre>typedef enum { MD_OK 0 /* correct execution MD_READY 0 /* ready MD_REJECT 1 /* not accepted (queue full or empty) MD_INAUGURATION 2 /* inauguration - possible inconsistency } MD_RESULT;</pre>

2.3.4.6.3 Procedure 'lm_send_request'

Definition	Adds the Link Header to a packet and insert it into the Send_Queue. If the link layer accepts a packet for transmission, it sets its status to MD_PENDING.	
Syntax	MD_RESULT	lm_send_request (ENUM8 bus_id UNSIGNED32 source, UNSIGNED32 destination, MD_PACKET * packet;)
Input	bus_id	Selects one of 16 different link layers.
	source	Device_Address of the source device. If 'source' is zero, the link layer includes into the packet its own Device_Address.
	destination	Device_Address of the destination device, or a broadcast address if set to the highest possible Device_Address.
	packet	Pointer to a data structure which contains the packet. The size and status of the packet are contained in the packet itself.
Return		any MD_RESULT

2.3.4.6.4 Procedure type 'LM_SEND_CONFIRM'

Definition	A procedure of that type is called by the link layer to return the packet to the pool when it has been transmitted or is no longer needed.	
Syntax	<pre>typedef void (* LM_SEND_CONFIRM) (MD_PACKET * packet);</pre>	
Input	packet	Pointer to a data structure which contains the packet sent or flushed. The size and status of the packet is contained in the packet itself.
Usage	1 - The actual procedure 'nm_get_pack' (of type LM_GET_PACK) has been previously subscribed in the 'lm_init' procedure. 2 - The link layer is expected to set the packet status either to MD_SENT (successful sending) or to MD_FLUSHED (queue flushed) before calling 'nm_send_confirm'.	

2.3.4.6.5 Procedure type 'LM_GET_PACK'

Definition	A procedure of this type is called to request a free packet from a pool. This procedure, if successful, supplies a single packet with the owner field set to the value of the owner parameter.	
Syntax	<pre>typedef void (* LM_GET_PACK) (void * * owner, MD_PACKET * * packet);</pre>	
Input	owner	Identifies the pool from which the packet is taken
	packet	Pointer to a data structure where the packet is stored.
Usage	1 - The actual procedure 'nm_get_pack' (of type LM_GET_PACK) has been previously subscribed in the 'lm_init' procedure. 2 - The link layer shall specify as owner only the pool which has been assigned to it at initialisation time.	

2.3.4.6.6 Procedure type 'LM_RECEIVE_INDICATE'

Definition	When a packet is received, the link layer shall call a procedure of this type.	
Syntax	<pre>typedef void (* LM_RECEIVE_INDICATE) (ENUM8 bus_id);</pre>	
Input	bus_id	Bus_Id (0..15)
Usage	1 - The actual procedure 'nm_receive_indicate' (of type LM_RECEIVE_INDICATE) has been previously subscribed in the lm_init procedure. 2 - This procedure may be NULL if polling ('lm_receive_poll') is used. 3 - This indication procedure is called from an interrupt service routine but it is allowed to make kernel calls. It is intended to wake up the Messenger.	

2.3.4.6.7 Procedure 'lm_receive_poll'

Definition	Reads one received packet out of the Receive_Queue and pass the packet reference to the network layer.	
Syntax	<pre>MD_RESULT lm_receive_poll (ENUM8 bus_id, unsigned * source, unsigned * destination, MD_PACKET * * packet);</pre>	
Input	bus_id	Bus_Id (0..15) of this link layer
Return	any MD_RESULT	
Output	source	Device_Address of the source device
	destination	Device_Address of the destination device or 'broadcast' address.
	packet	Pointer to a data structure which contains the received packet, or NULL if the queue is empty. The size of the packet is contained in the packet itself.
Usage	1 -The packet descriptor can be reused after this procedure is called. 2 -If the result of this procedure is MD_OK, the Receive_Queue is emptied by one position, and the other arguments of this procedure become significant. 3 -If this procedure is called when the Receive_Queue is void, it returns MD_REJECT as a result and NULL as 'packet'.	

2.3.4.6.8 Procedure type 'LM_STATUS_INDICATE'

Definition	When an exception occurs, the link layer shall call a procedure of this type to signal the fact.	
Syntax	<pre>typedef void (* LM_STATUS_INDICATE) (ENUM8 bus_id, MD_RESULT status);</pre>	
Input	bus_id	Bus_Id (0..15)
	status	MD_OK MD_REJECT (bus not available) MD_INAUGURATION (train bus undergoes inauguration)
Usage	1 - The actual procedure 'nm_status_indicate' (of type LM_STATUS_INDICATE) has been previously subscribed in the 'lm_init' procedure. 2 - The status, of type MD_RESULT', is an input parameter.	

2.3.4.6.9 Procedure 'lm_get_status'

Definition	Retrieves detailed information on the link layer.	
Syntax	<pre>MD_RESULT lm_get_status (ENUM8 bus_id, BITSET8 selector, BITSET8 reset, BITSET8 * status);</pre>	
Input	bus_id	Bus_Id (0..15)
	selector	selects the interesting bits returned in status Each piece of information is represented by a single bit which is set by the link layer and reset by the service user. The following three bits are defined MD_RECEIVE_ACTIVE = 1set whenever a correct frame is received from the bus. MD_SEND_ACTIVE = 2set when a frame has been transmitted. MD_RECEIVE_OVERFLOW = 4set when a received frame is lost because there is no free packet.
	reset	selects the bits which are to be cleared afterwards.
Return		any MD_RESULT
Output	status	returns the value of the selected status bits.
Usage	The network layer is expected to reset the MD_SEND_ACTIVE bit when reading the status of a vehicle bus link layer.	

2.3.4.6.10 Procedure 'lm_send_queue_flush'

Definition	Flushes the Send_Queue in the link layer and sets the packet status to MD_FLUSHED. Calls nm_send_confirm once for each packet which has been inserted by lm_send_request and which has not yet been sent.	
Syntax	<pre>MD_RESULT lm_send_queue_flush (ENUM8 bus_id /* optional */);</pre>	
Input	bus_id	Bus_Id (0..15)
Return		any MD_RESULT

2.3.4.6.11 Procedure 'lm_init'

Definition	Initialises the link layer, flushes the Send_Queue and subscribes the indication procedures from the network layer.	
Syntax	<pre>MD_RESULT lm_init ENUM8 (LM_RECEIVE_INDICATE bus_id, LM_GET_PACK nm_receive_indicate, void * * nm_get_pack, LM_SEND_CONFIRM owner, LM_STATUS_INDICATE nm_send_confirm, nm_status_indicate);</pre>	
Input	bus_id	Bus_Id (0..15)
	nm_receive_indicate	procedure of the network layer which the link layer calls each time it receives a packet.
	get_packet	procedure of the packet pool which the link layer calls each time it needs a void packet.
	owner	packet pool the link layer use to get packets.
	put_pack	procedure of the packet pool which the link layer calls to dispose of a packet.
	nm_status_indicate	procedure of the network layer which the link layer calls to signal an exception (e.g. inauguration).
Return		any MD_RESULT

2.3.4.6.12 Procedure 'lm_get_dev_address'

Definition	reads the own Device_Address corresponding to the specified link layer.	
Syntax	<pre>MD_RESULT lm_get_dev_address (ENUM8 bus_id, UNSIGNED * device_address);</pre>	
Input	bus_id	Bus_Id (0..15)
Return		any MD_RESULT
Output	device_address	Device_Address of that device over this link layer
Usage	On a train bus link layer, the Device_Address is retrieved from which the Node_Address can be deduced.	

2.3.5 Message Network Layer**2.3.5.1 Purpose**

The network layer relays packets, as illustrated in Figure 23, either:

- from the transport layer of its station to a link layer (outbound packets);
- from one of its link layer to the transport layer (inbound packets);
- from one link layer to another link layer in a router Node (transit packets).

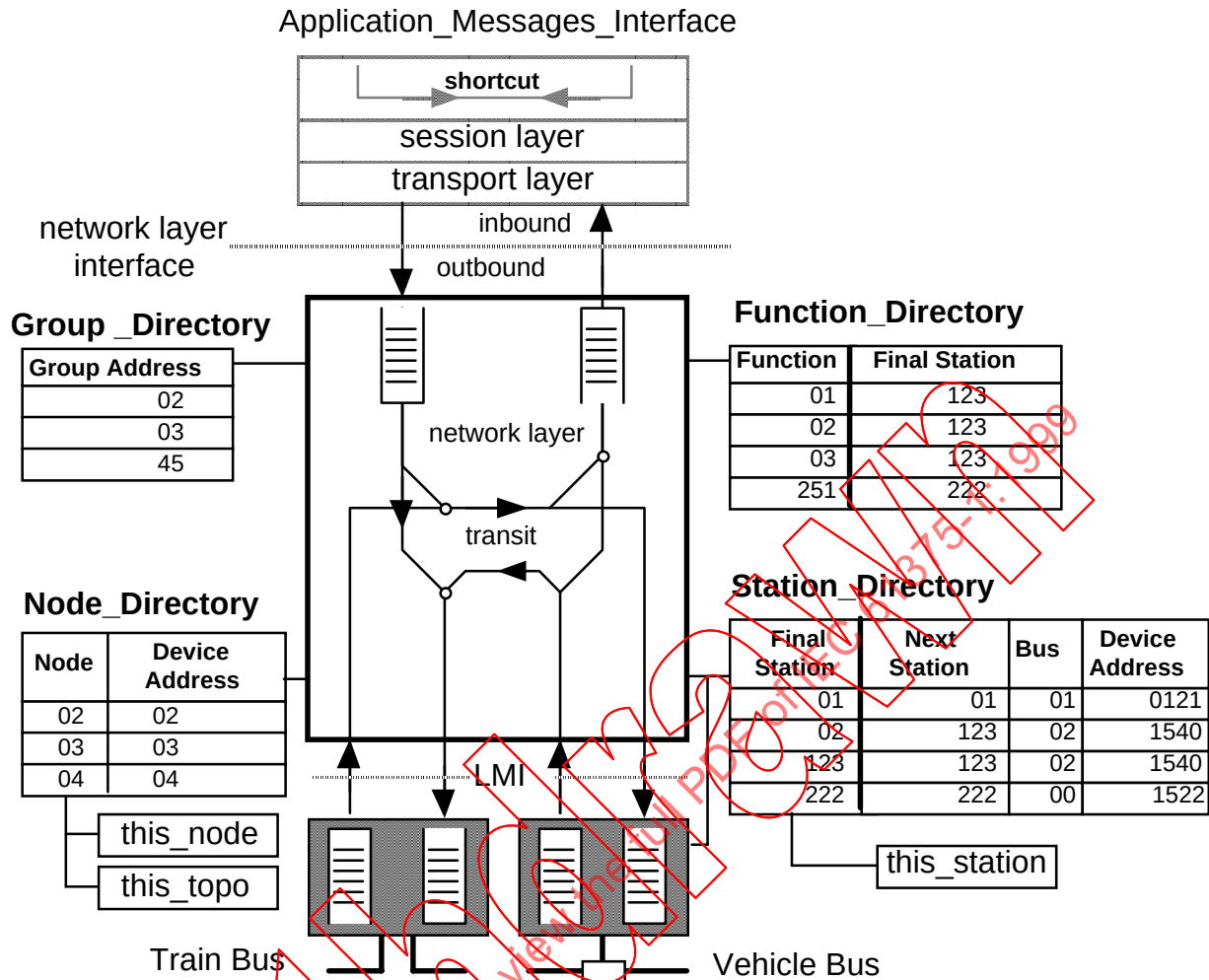


Figure 23 - Network layer on a Node

The network layer routes packets from an Origin station to a Final station.

To this effect, the network layer uses the mapping provided by several directories:

- the station directory,
- the function directory,
- the group directory, and
- the node directory.

The network layer is connectionless.

2.3.5.2 Directories

2.3.5.2.1 Station directory (option)

The Network Layer shall map a station identifier to the link layer address of that station and vice-versa.

In a terminal station, this mapping may be obtained by extending the Station_Id with a fixed leading field to obtain the Device_Address (simple mapping).

EXAMPLE - The 8-bit Station_Id '11111101'B may be mapped to the 12-bit Device_Address '000011111101'B on the MVB.

On a Router station, the mapping shall be implemented by a station directory structured as follows:

Station_Id	station identifier (key to the station directory)
Next_Station_Id	station identifier of the next station over which the station can be reached.
Bus_Id	link layer over which Next_Station can be reached
Device_Address	Device_Address on the bus where Next_Station can be reached

NOTE 1 The station directory allows to build the Link_Header of an outgoing packet, but it is also used to identify the source link layer address of an incoming packet. The reason for this is that the Transport Layer should not handle bus-specific addresses of arbitrary size, but only station identifiers.

NOTE 2 Station_Id and Next_Station are identical when the station can be directly reached, they are distinct when packets are routed between two vehicle busses or between a vehicle bus and a sensor bus.

NOTE 3 The access primitives of the station directory are defined in the Application layer Interface (AMI)

2.3.5.2.2 Function directory

The Network Layer shall implement the function directory, which maps a Function_Id to a Station_Id, structured as follows:

Function_Id	function identifier (key to the function directory)
Station_Id	station identifier (key to the station directory)

NOTE 1 The function directory belongs to the network layer, but it can be accessed by all Layers, except the link layer.

NOTE 2 The access primitives of the function directory are defined in the Application layer Interface (AMI)

2.3.5.2.3 Group directory

The Network Layer of a station participating in multicast communication shall indicate to which Group this station pertains through a group directory, structured as follows:

Membership	list of Groups

NOTE The access primitives of the group directory are defined in the Application layer Interface (AMI)

2.3.5.2.4 Node directory (option)

The Network Layer of a Node shall map the Node_Address to the Device_Address on the train bus.

When the WTB is used as a train bus, a simple mapping shall be used, obtained by prefixing the six-bit Node_Address by two leading '0' to form the eight-bit WTB Device_Address. The node directory is read-only since its contents are defined by the inauguration.

In fixed train compositions where a simple mapping is not adequate, the Node Address shall be mapped to the Device_Address by using a node directory, structured as follows:

Node_Address	Node_Address (key to the node directory)
Bus_Id	link layer corresponding to the train bus
Device_Address	bus-dependent device address

NOTE The access primitives of the node directory are defined in the Application layer Interface (AMI).

2.3.5.3 Network layer constants and variables

Distinguished values of Station_Id or Next_Station:

Station_Id =	Meaning
AM_SAME_STATION	0 (constant)
AM_UNKNOWN	255 (constant)
this_station	8-bit station identifier of this station. If a station has no allocated identifier, this_station = AM_UNKNOWN.
final_station	station indicated in the final Network_Address.
origin_station	station indicated in the origin Network_Address.
next_station	station over which the network layer sends a packet or from which it received it.

Distinguished values of Node (all these values are UNSIGNED6):

Node =	Meaning
AM_SAME_NODE	0 (constant)
this_node	the 6-bit Node_Address of this Node
AM_ANY_TOPO	0 (constant) any Topo_Counter value
origin_node	the 6-bit Node_Address in the origin Network_Address
final_node	the 6-bit Node_Address in the final Network_Address.
this_topo	the value of the Topo_Counter on this station, registered by am_set_current_tc.
my_topo	the value of the Topo_Counter indicated by the application for this conversation
packet_topo	the value of the Topo_Counter carried in the packet

Miscellaneous procedures:

multicast	returns true if Group addressing is used
member (group)	returns true if Node belongs to the Group and if multicast is used
fundi ()	return the Station_Id indicated by the Function_Directory
stadi ()	return the Link_Address indicated by the Station_Directory.

2.3.5.4 Network_Address

2.3.5.4.1 Format

The network layer receives from its transport layer or from one of the link layers a Final Network_Address with each packet.

The network layer uses this Final Network Address to build the Link_Header and the Network_Header of a packet it forwards.

The network layer reads the Link_Header and the Network_Header of an inbound packet and converts them into an origin Network_Address to the transport layer.

A convenient encoding of the final and origin Network_Address is shown in Figure 24.

0	1	2	3	4	5	6	7
fsu	fgi	final_node_or_group					
final_function_or_station							
next_station							
frv	ftv	my_topo					

Final Network Address

0	1	2	3	4	5	6	7
osu	ogi	origin_node					
origin_function_or_station							
this_station							
orv	otv	this_topo					

Origin Network Address

NOTE The figure is not a transmission format, but an interface format.

Figure 24 - Encoding of the Network_Address

2.3.5.4.2 System or User (fsu/osu)

The most significant bit of the first octet of a Network_Address, 'fsu' or 'osu' specifies, if it is:

- 1: a System_Address: the octet identifies a station, or if it is
- 0: a User_Address: the next octet identifies a function.

The 'osu' bit in the Origin_Address shall have the same value as the 'fsu' bit in the Final_Address.

2.3.5.4.3 Group or Individual (fgi/ogi)

The second most significant bit of the first octet of the final Network_Address specifies, if it is:

- 1: a Group address or, if it is
- 0: an Individual address

The 'fgi' and 'ogi' bit shall be the same in the Origin_Address and in the Final_Address.

2.3.5.4.4 Node or Group

The six bits 'origin_node' represent

- a single Node_Address of the origin node.

The six bits 'final_node_or_group' represent:

- a Group_Address, if the 'fgi' bit specifies a Group,
- a single Node_Address otherwise.

2.3.5.4.5 Function or station

The second octet of a Network_Address contains:

- if 'fsu'/'osu' is 'System', a station identifier;
- if 'fsu'/'osu' is 'User', a function identifier.

2.3.5.4.6 Link_Address

The third octet, 'next_station', identifies the station attached to the same Node to which a packet is to be sent or from which a packet has been received (it is not necessarily the origin station or the final station).

The network layer obtains the corresponding Link_Address (Bus_Id and Device_Address) from its Station_Directory, or, for an inbound packet, can deduce Station_Id of Next_Station from the Device_Address and Bus_Id.

'next_station' may also indicate the own station (AM_SAME_STATION) or an unknown station (AM_UNKNOWN).

A Node sending over the train bus has no defined next_station (AM_UNKNOWN), but if the Node uses a node directory, Next_Station is the entry to that directory.

2.3.5.4.7 Topography counter

The fourth octet of the final and origin Network_Address contains the Topo_Counter.

The most significant bit of that octet, 'frv' resp. 'orv', is '0'.

The second most significant ('ftv', resp. 'otv') indicates that the least significant six bits contain a valid counter value (except if a group address is used).

2.3.5.5 Network Header encoding

2.3.5.5.1 Outbound and inbound packets

The final Network_Address and origin Network_Address of an outbound packet is used to build the Link_Header and the Network_Header fields on the MVB (Topo_Counter is not used), as shown in Figure 25.

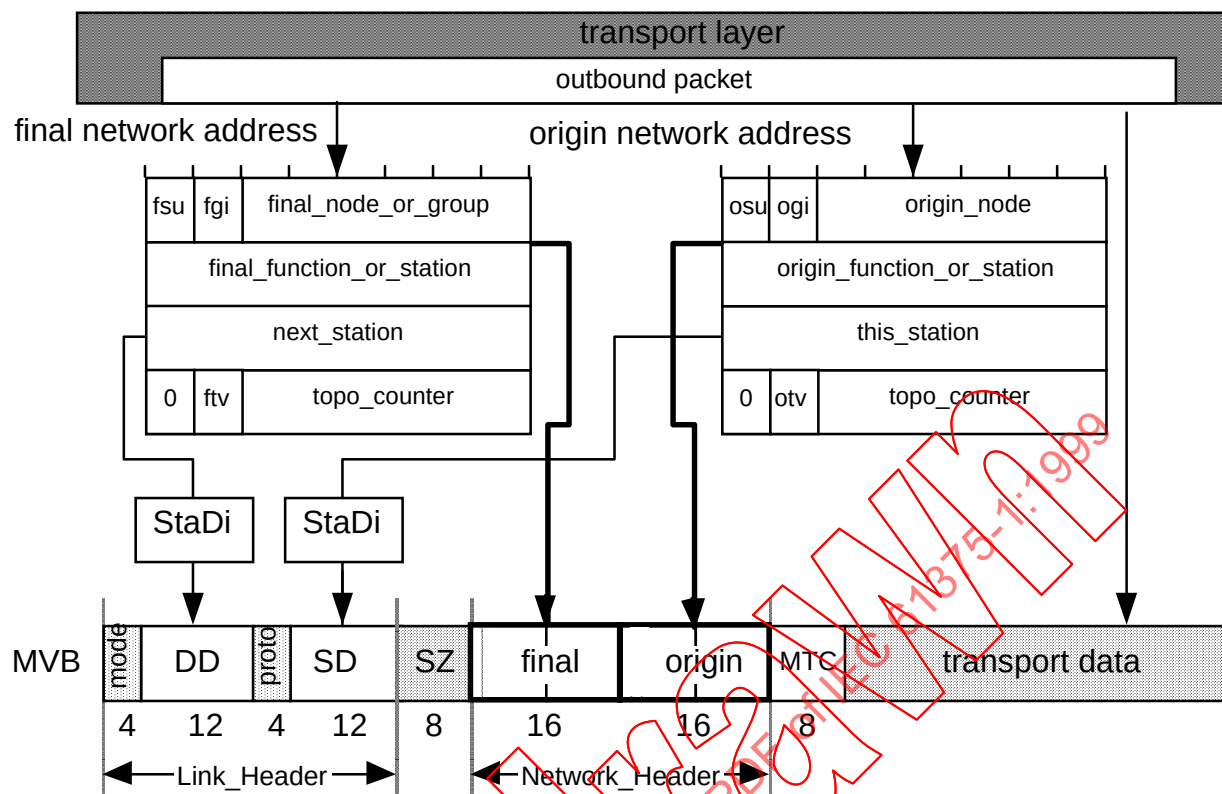


Figure 25 - Building of the addresses in an outbound packet

2.3.5.5.2 Network header encoding on the vehicle bus

The network header on the vehicle bus shall be encoded according to the following specification, illustrated by Figure 26 for the MVB:

Network_Header ::= RECORD

```

{
  fsu                      ENUM1          -- final address (system or
                                           user)
  {
    USER (0)              -- user address (function) is
                           used
    SYSTEM (1)            -- system address (station) is
                           used
  },
  fgi                      UNSIGNED1,      -- group, considered together
                                           with 'tpv'
  ONE_OF [fgi * 2 + ogi]
  {
    ['00'B] final_node UNSIGNED6          -- vehicle internal
                                           (node = 0)
    ['01'B] final_node UNSIGNED6          -- outgoing (sent to node)
                                           (node <> 0)
    ['10'B] final_topo UNSIGNED6,         -- incoming: Topo_Counter of
                                           node (if <> 0)
    ['11'B] final_group UNSIGNED6         -- group address for multicast.
  },
  ONE_OF [fsu]                -- depending on system or user
                               address
  {
    [USER] final_function,  UNSIGNED8,
    [SYSTEM] final_station UNSIGNED8
  }
  osu                      ENUM1,          -- same as 'fsu', reject
                                           otherwise.
  ogi                      UNSIGNED1,      -- considered together with gni
  ONE_OF [fgi * 2 + ogi]      -- same as above
  {
    ['00'B] origin_node UNSIGNED6,        -- vehicle internal
                                           communication (node = 0)
    ['01'B] origin_topo UNSIGNED6         -- outgoing: Topo_Counter of
                                           station (if <>0)
    ['10'B] origin_node UNSIGNED6,        -- incoming: origin node
                                           (if <>0)
    ['11'B] origin_topo UNSIGNED6,        -- origin node for multicast
  },
  ONE_OF [fsu]
  {
    [USER] origin_function,  UNSIGNED8,
    [SYSTEM] origin_station UNSIGNED8
  }
}

```

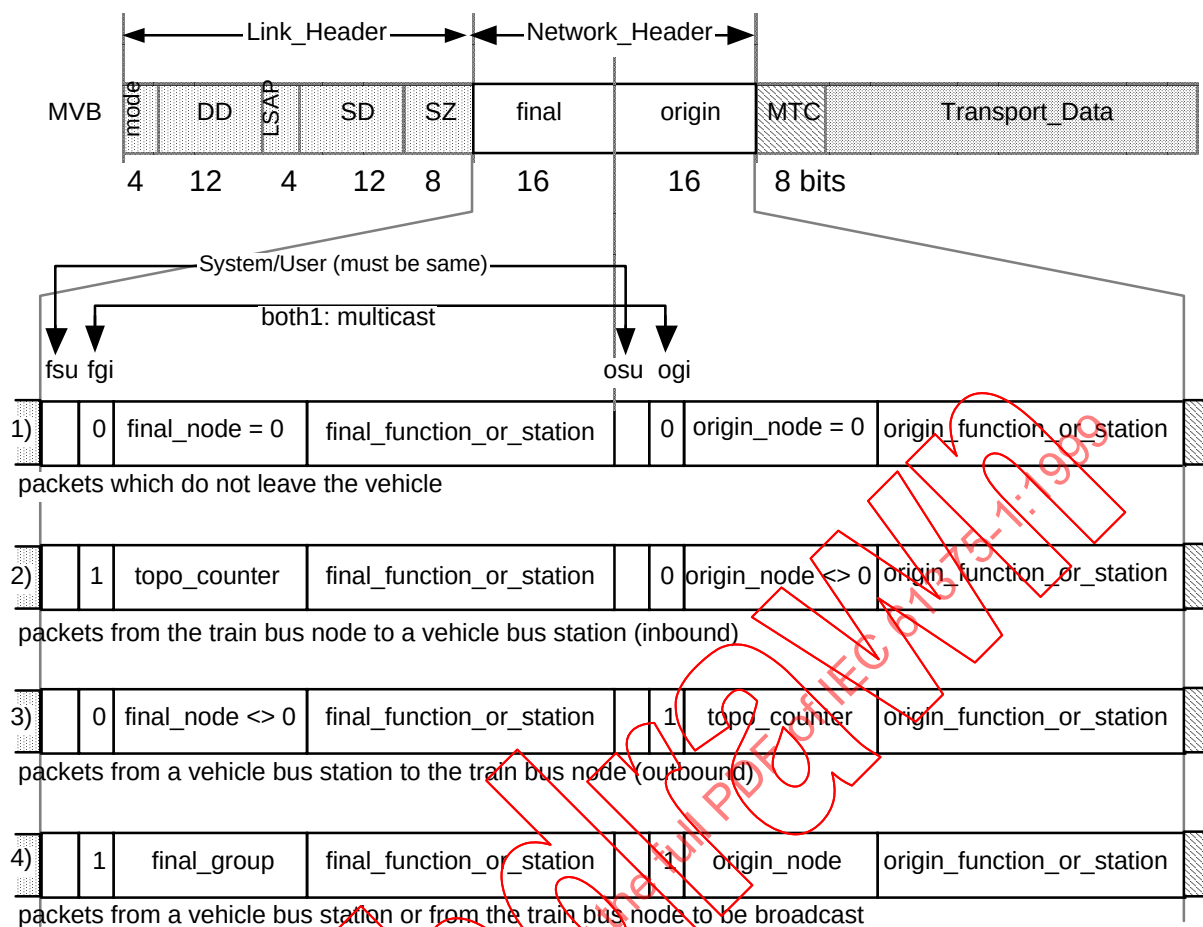



Figure 26 - Network address encoding on the vehicle bus (example MVB)

The meaning of the network address fields differ in the following four situations as shown in Figure 26:

- packets sent within a vehicle bus only (final_node = origin_node = AM_SAME_NODE) carry no topo_counter: the 'fgi' and 'ogi' shall be both '0';
- packets sent by the Node with (origin_node <> AM_SAME_NODE) carry 'topo_counter' instead of final_node: the 'fgi' bit shall be '1';
- packets sent to the Node with (final_node <> AM_SAME_NODE) carry 'topo_counter' instead of origin_node: the 'ogi' bit shall be '1';
- packets sent in multicast from a vehicle bus station to the Node or vice-versa: the 'fgi' and 'ogi' shall be both '1'.

If the packet sent from a station is to be broadcast over the train bus only, the link layer address shall be that of the Node, 'origin_node' shall be 'AM_SAME_NODE'.

2.3.5.5.3 Network header encoding on the train bus

The network header on the train bus shall be encoded according to the following specification, illustrated by Figure 27 for the WTB:

```

Network_Header ::= RECORD
{
  fsu                                ENUM1
  {
    USER                            (0)          -- user address (function) is
                                                used
    SYSTEM                          (1)          -- system address (station) is
                                                used
  },
  fgi                                UNSIGNED1,      -- 1 if Group, 0 if Individual
  ONE_OF [fgi]
  {
    [0] final_node                   UNSIGNED6,
    [1] final_group                  UNSIGNED6
  },
  ONE_OF [fsu]
  {
    [USER] final_function,           UNSIGNED8,
    [SYSTEM] final_station           UNSIGNED8
  },
  osu                                ENUM1,          -- same as 'fsu', if different:
                                                reserved
  ogi                                UNSIGNED1,      -- same as fgi ( 1 = multicast)
  origin_node                        UNSIGNED6,      -- origin is always a single
                                                node
  ONE_OF [snu]
  {
    [USER] origin_function,          UNSIGNED8,      -- if this is a user address
    [SYSTEM] origin_station          UNSIGNED8      -- if this is a system address
  }
}

```

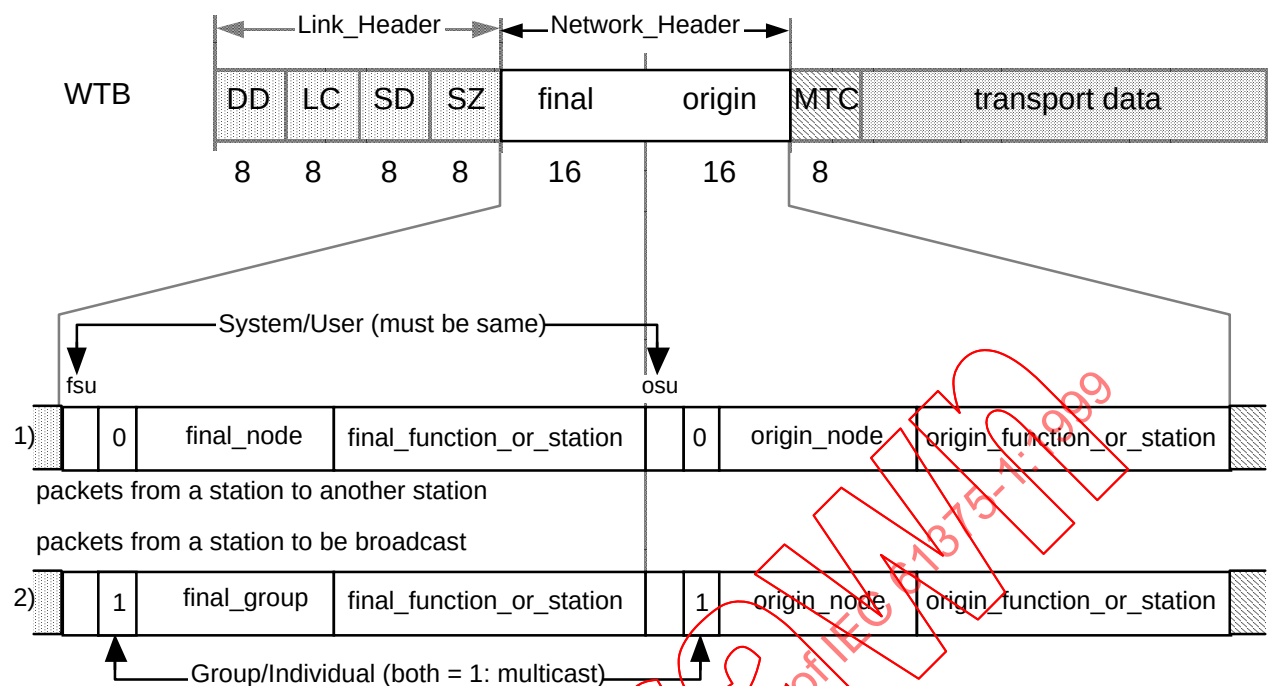


Figure 27 - Network address encoding on the train bus (example with WTB)

The Origin_Node and the Final_Node (or Group) shall be defined (<> AM_UNKNOWN)

Both the Final_Address and the Origin_Address shall use the same type of addressing (system or user), the 'snu' bits shall be set to the same value.

The Origin_Address shall be an individual Node_Address.

If the final address specifies a Group (multicast), the 'fgi' and 'ogi' bits shall both be '1'.

2.3.5.6 Routing in the network layer

2.3.5.6.1 Situations

The nine situations listed in Table 9 define the behaviour of a full-fledged network layer connected to:

- a transport layer;
- one or more vehicle bus link layer;
- a train bus link layer.

Table 9 - Routing situations

Situation	From	To	Remarks
1	this transport layer	this transport layer	shortcut within same device
2	this transport layer	any vehicle bus	there may be several vehicle busses
3	this transport layer	the train bus	station is Node
4	any vehicle bus	this transport layer	there may be several vehicle busses
5	any vehicle bus	Another vehicle bus	router function
6	any vehicle bus	the train bus	router function
7	the train bus	this transport layer	station is Node
8	the train bus	any vehicle bus	router function
9	the train bus	the train bus	not foreseen

A terminal station attached to the vehicle bus encounters only situations 1, 2 and 4.

A terminal station attached to the train bus encounters only situations 1, 3 and 7.

A Router station attached to two vehicle busses encounters only situations 1, 2, 4 and 5.

2.3.5.6.2 Return path checking

The network layer shall ensure that all packets belonging to a Call_Message and to its corresponding Reply_Message take the same route.

The network layer shall check for each packet it receives that it is possible to send back a packet to the Origin_Address, otherwise the network layer shall not forward the packet.

This is checked assuming that a packet is received for which the Origin_Address is exchanged with the Final_Address and the source address is exchanged with the destination address and by checking that Next_Station is defined in that case.

At initialisation time, 'next_station' may be undefined, in particular when the calling station has no entry in the station directory.

If a system message is received from a station not included in its station directory, the network layer shall assume that the origin station in the Origin_Address has the Link_Address corresponding to the source device and enter it into the station directory. A user message would be rejected under these conditions.

NOTE This implicit entry can be corrected afterwards if necessary by management messages.

2.3.5.6.3 Topography consistency

To safeguard against configuration changes of the train bus during a message transfer, packets sent to or from a Node over the vehicle bus contain the Topo_Counter.

The network layer shall hold the most recent value of the Topo_Counter in its variable 'this_topo'.

NOTE 1 The value of 'this_topo' is communicated to the network layer by the AMI service 'am_set_current_tc'.

NOTE 2 If the station is a Node, 'this_topo' is incremented when an inauguration takes place.

NOTE 3 When making a call, an Application indicates the value 'my_topo'. If the application does not know the Topography or does not care about consistency, it sets my_topo = AM_ANY_TOPO.

The transport layer communicates 'my_topo' to the network layer as part of the Network_Address.

The network layer on a Node shall compare 'this_topo' with the value found in the incoming packets, called 'packet_topo'.

When a Node receives a packet from its vehicle bus:

- with the packet_topo in the Origin_Address equal to its this_topo or to AM_ANY_TOPO, the Node shall insert its own Node_Address (this_node) in the Origin_Address and forward the packet over the train bus.
- with the packet_topo in its Origin_Address different from this_topo, but different from AM_ANY_TOPO, the Node shall cancel the transfer as explained in 2.3.5.6.4.

When a Node receives a packet from another Node over the train bus:

- a packet directed to one of its stations or functions, it shall insert 'this_topo' in the Final_Address as packet_topo;
- the final station will check that the packet_topo sent by the Node is identical to this_topo. Otherwise, the final station shall cancel the transfer as explained in 2.3.5.6.4.

2.3.5.6.4 Cancelling

To cancel a transfer, the network layer shall forward the infringing packet to its transport layer, which will send to the origin of the packet a Disconnect_Request with reason AM_INAUG_ERR.

During inauguration, a node shall respond to any packet coming from the vehicle bus and addressed to another vehicle by a Disconnect_Request with reason AM_INAUG_ERR.

When a device receives a Disconnect_Request with reason AM_INAUG_ERR, it shall cancel all its ongoing connections over the train bus.

NOTE A connectionless network layer is not supposed to act on the protocol, so it delegates this task to the transport layer. The only situation in which this occurs is because of an inauguration error.

2.3.5.6.5 Routing algorithms

A Router shall forward a packet as specified:

- for outbound packets in Table 10,
- for inbound and transit packets from a vehicle bus in Table 11, and
- for inbound and transit packets coming from a train bus in Table 12.

Table 10 - Routing of packets coming from the transport layer

From	To	Condition
this transport layer	Any	The transport layer communicates next_station and the Topo_Counter to the network layer in the Network_Address. The transport layer does not check the Topo_Counter of outbound packets.
(1)	Transport layer	In single cast: not possible since the Session layer shortcuts the network when the partner resides on the same station. In multicast: a broadcast packet is transmitted by this station may be addressed to a function residing on this station as a result of loop-back from the link layer when: (member AND (fundl(function_id) = this_station))
(2)	Vehicle bus link layer	(final_node = AM_SAME_NODE) AND (next_station <> AM_SAME_STATION) AND (next_station <> this_station) AND (Link_Address of next_station = defined)
(3)	train bus link layer	(next_station = AM_SAME_STATION) OR ((next_station = this_station) AND (final_node <> AM_SAME_NODE)) This situation only exists on a Node. The transport layer indicates to forward this packet over the train bus by specifying: (next_station = AM_SAME_STATION). The transport layer is supposed to check that there is a train bus connection (e.g. no inauguration in progress).

Table 11 - Routing of packets coming from a vehicle bus

From	To	Condition
4	Any	For all packets, the network layer determines next_station by analysing the final network address: IF (final_node <> AM_SAME_NODE) THEN next_station = fundi(AM_ROUTER_FCT) ELSE IF User THEN next_station = fundi(function_id) ELSE next_station = final_station END next_station = stadi (station_id)
	transport layer	All cases not covered by the other conditions. Packet are not sent to the train bus, no station would receive it. The network layer communicates the Link_Address of the source device to the transport layer in next_station. The transport layer is expected to send a Disconnect_Request if the partner does not exist.
	another vehicle bus	((final_node = AM_SAME_NODE) OR (final_node = this_node)) AND ((packet_topo = this_topo) OR (packet_topo = AM_ANY_TOPO)) AND ((final_station <> this_station) AND (final_station <> AM_SAME_NODE) AND (final_station <> AM_UNKNOWN)) Even if a packet is not sent over the train bus, its topography will be checked if (final_node <> AM_SAME_NODE).
6	train bus	((final_node <> this_node) AND (final_node <> AM_SAME_NODE) AND ((packet_topo = this_topo) OR (packet_topo = AM_ANY_TOPO))) OR member. A packet is sent over the train bus only if its packet_topo is correct or if multicast is used.

Table 12 - Routing of packets coming from the train bus

From	To	Condition
Train bus		The main difference with packets coming from a vehicle bus is the handling of the Topo_Counter. In packets coming from the train bus, the Node inserts its this_topo in place of final_node so that the final station can check it.
7	transport layer	((final_node = this_node) OR (member)) AND ((final_station = this_station) OR (final_station = AM_SAME_STATION) OR (final_station = AM_UNKNOWN)) The packet is intended for this Node and no other station would accept it. The transport layer decides if the partner exists.
8	vehicle bus	((final_node = this_node) OR (member)) AND ((final_station <> this_station) AND (final_station <> AM_SAME_STATION) AND (final_station <> AM_UNKNOWN)) There is a station attached to this Node which would accept the packet.
9	train bus	This situation is not foreseen.

2.3.5.7 Network layer interface

The network layer interface to the transport layer is not specified and it is not open.

The application may set up and consult the directories, which are conceptually part of the network layer, through the Application_Messages_Interface.

2.3.6 Message transport layer

2.3.6.1 Purpose

This subclause describes two transport protocols:

- the Message Transport Protocol (MTP) used for point-to-point communication;
- the (optional) Multicast Protocol (MCP) used for multicast communication.

2.3.6.2 Message Transport Protocol

The transport layer transfers a message from a Producer to one Consumer, and provides the following services:

- a) segmentation of long messages into fixed-size packets for transmission;
- b) flow control and error recovery from end to end through a sliding window protocol;
- c) cancelling.

The MTP opens a connection for the duration of one message and in one direction only.

The MTP supports simultaneous transport of several messages between any two Applications Processes.

The MTP supports simultaneous transport of several messages by the same Application. Each connection is distinguished by a Session layer reference parameter 'session_ref'.

However, the MTP supports only one connection at the same time and in the same direction between the same two Producers and Consumers.

NOTE The terms 'Producer' and 'Consumer' refer to the sender and receiver of a message at the transport layer. The terms 'origin' and 'final' refer to the end partners over the network. The terms 'source' and 'destination' to the Device_Address of partners over the same bus.

2.3.6.3 Packet exchange

The transport of a message shall be divided into three phases:

- connection establishment;
- acknowledged data transfer;
- disconnection.

EXAMPLE - Figure 28 shows a simple packet exchange with a window size of 1.

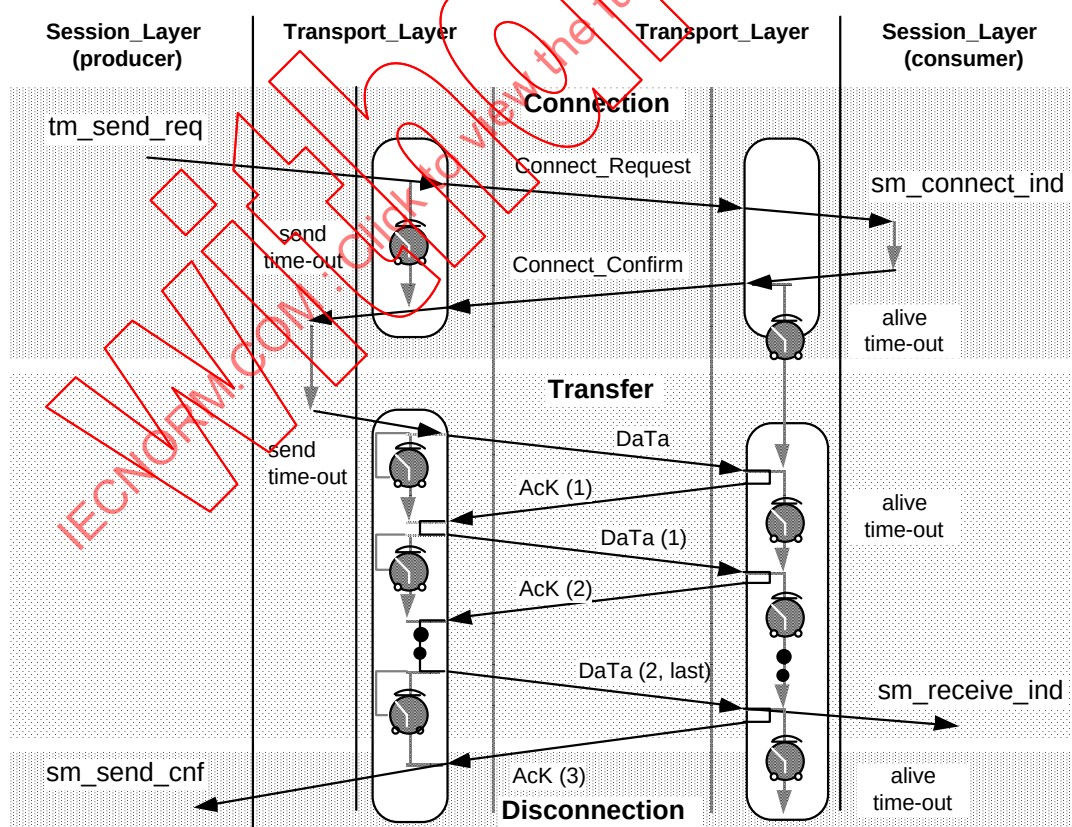


Figure 28 - Transport packet exchange

2.3.6.3.1 Connection establishment

The Producer requests the Consumer to accept a message by sending a Connect_Request packet. This packet contains the total length of the message, the size of the packets, the window size and a Connection Reference supplied by the transport layer which uniquely identifies the Connect_Request in case this one needs to be repeated.

The Consumer answers a Connect_Request packet either through a Connect_Confirm if it accepts the message or through a Disconnect_Request packet if it rejects it.

The window size is negotiated during connection establishment: the Producer proposes a credit and the Consumer responds with its accepted credit, which may not be higher than the proposed credit.

The same holds for the packet size. The Connect_Request packet has the smallest possible size in the network, which is dictated by the 256 bits an MVB can transmit.

The credit and packet size which the Consumer returns holds for all subsequent packets of the same message.

The Producer may insert up to 14 octets of data into the Connect_Request packet.

A message smaller or equal to 14 octets may be entirely delivered to the Consumer without DATA packets. The Connect_Confirm packet then acknowledges the reception of the whole message.

If Connect_Request is not confirmed or cancelled, the procedure shall attempt retransmission up to three times before disconnecting.

2.3.6.3.2 Acknowledged data transfer

The Producer sends the individual data packets of the message to the Consumer. It may send at most AcpCredit (Accepted Credit) packets without receiving an acknowledgement for them.

The Consumer shall acknowledge positively the data packets by an Acknowledgement packet indicating the number of the next packet it expects.

The Consumer side may bundle acknowledgements, i.e. acknowledge several packets at the same time by acknowledging the packet with the highest sequence number only.

The Producer shall be able to handle bundled acknowledgements.

When packets cease to be acknowledged, the transport layer shall retransmit them up to a maximum of three times before disconnecting.

The Consumer may indicate to the Producer that it received an out-of-sequence packet. In this case, it shall send a Negative_Acknowledgement packet indicating from which packet on it requires retransmission.

2.3.6.3.3 Disconnection

Disconnection is implicit if transfer is not explicitly cancelled. The connection is however not immediately severed since late packets can still arrive if acknowledgements get lost.

Disconnection is explicit when either the Producer or the Consumer sends a Disconnect_Request packet.

Either party which receives a Disconnect_Request (except in response to a Connect_Request or to a Disconnect_Request with reason AM_REM_CANC) answers with a Disconnect_Confirm packet.

As an exception to this rule, a router station can send a Disconnect_Request if it cannot forward packets properly, for instance as a consequence of a change of composition.

NOTE Since a network layer is not supposed to act on the protocol, this packet is forwarded to the transport layer of the router, which sends the Disconnect_Request.

2.3.6.4 Connection reference

Each Connect_Request shall be identified by a Connection_Reference parameter.

The Connection_Reference shall be a 16-bit number initialised to an arbitrary value at start-up (e.g. taken from the real-time clock).

The transport layer shall increment the Connection_Reference by 1 for each new outgoing connection on that station.

NOTE 1 The Connection_Reference distinguishes the repetition of a Connect_Request (if a transmission error occurred) from a new different request.

NOTE 2 The Connection_Reference is called conn_ref in the state transition tables.

2.3.6.5 Transport layer packets

2.3.6.5.1 Packet types

The transport layer shall operate with the seven packet types illustrated in Figure 29 and specified in the following subclauses.

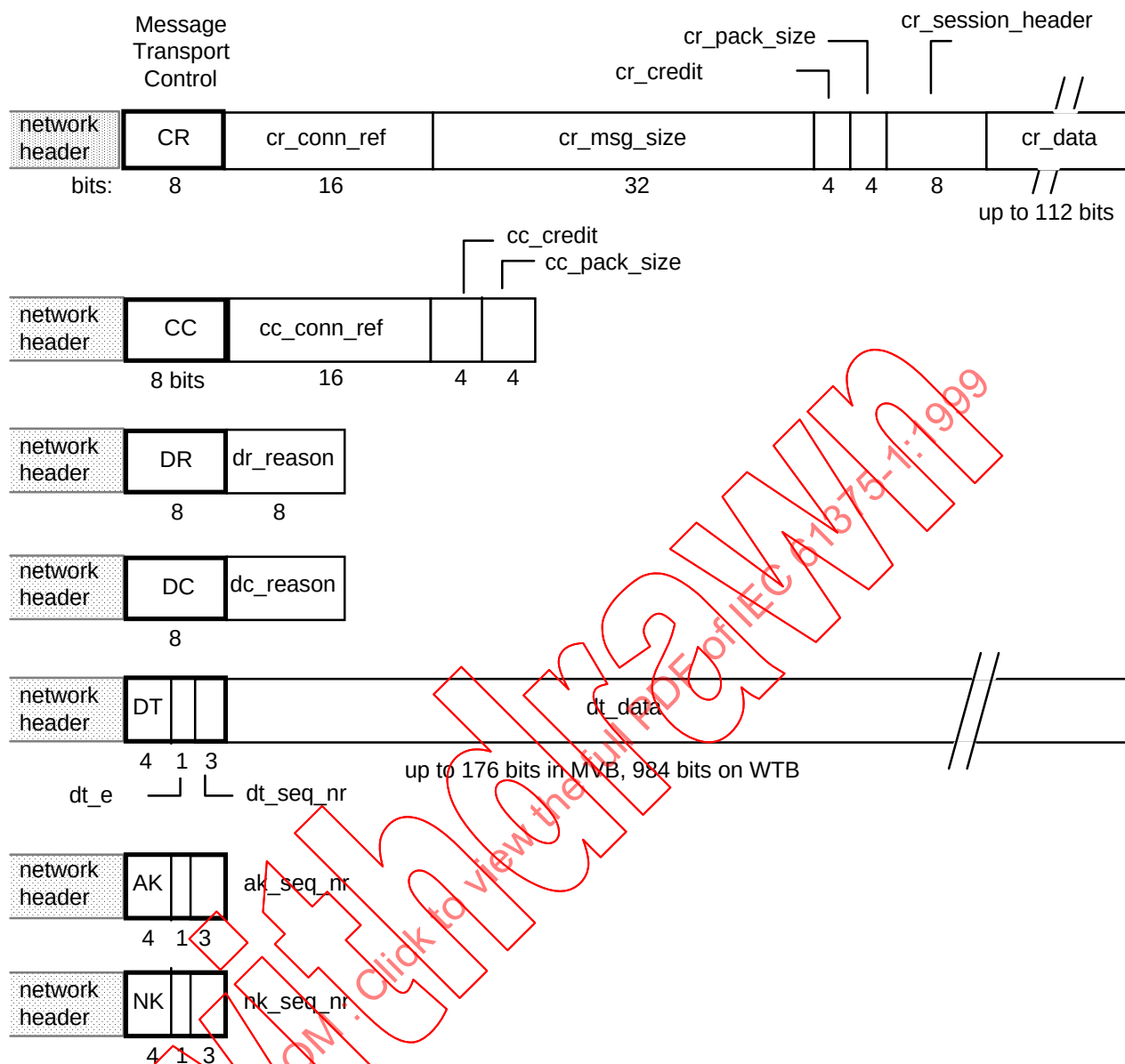


Figure 29 - Packet formats (transport layer body)

NOTE 1 The network header is specified in the Network Layer specification.

NOTE 2 The link header is specified in the link layer specification (it is bus-dependent).

2.3.6.5.2 Message Transport Control encoding

The Message Transport Control field shall be encoded as specified in Table 13.

Table 13 - Message Transport Control encoding

Packet Type	Packet	Short	Encoding							
	Connect_Request	CR	cl	0	0	0	0	0	0	0
Control (single-cast)	Connect_Confirm	CC	cl	1	0	0	0	0	0	1
	Disconnect_Request	DR	cl	co	0	0	0	0	1	0
	Disconnect_Confirm	DC	cl	co	0	0	0	0	1	1
	Broadcast_Connect	BC	1	0	0	0	1	0	bc_rept	
Multicast	Broadcast_Data	BD	1	0	0	0	1	1	0	0
	Broadcast_Repeat	BR	1	1	0	0	1	1	0	1
	Broadcast_Stop	BS	1	co	0	0	1	1	1	0
	Data	DT	cl	0	0	1	dt_e	dt_seq_nr		
Information	Acknowledgement	AK	cl	1	1	0	ak_e	ak_seq_nr		
	Negative Acknowledgement	NK	cl	1	1	1	nk_e	nk_seq_nr		

The first bit in the control field (cl) shall distinguish a Call (1) from a Reply (0) message.

The second bit (co) shall distinguish the origin as Consumer (1) or Producer (0).

The 'dt_e' bit (End-Of-Transfer) shall be set to '1' in the last packet of a message.

Undefined combinations are reserved.

NOTE 'CL' is conceptually part of the session layer, but it is used for the transport layer as well.

The result of a transfer shall be encoded in a packet by a field of type `Am_Result`:

NOTE The enumerated type `Am_Result` corresponds to the parameter `AM_RESULT` which is returned by the procedures of the application layer, but also by procedures of the session layer and transport layer. `AM_RESULT` is the type of a parameter in the 'C'-syntax, while `Am_Result` defines the encoding of that type for transmission.

The Packets shall be formatted as specified in the following:

[illegible]

```

        CONSUMER (0),
        PRODUCER (1)
    },
    packet_kind          ENUM2
    {
        CONTROL (0)
        DATA (1)
        ACK (2)
        NAK (3)
    ONE_OF [packet_type]
    {
        [CONTROL]          Control_Packet
        [DATA]              Data_Packet,
        [ACK]               Ack_Packet,
        [NAK]               Nak_Packet
    }
}

Control_Packet ::= RECORD
{
    command ::=          ENUM4
    {
        CR (0),
        CC (1),
        DR (2),
        DC (3)
        BC1 (8)
        BC2 (9)
        BC3 (10)
        BS (12)
        BR (13)
        BD (14)
    },
    ONE_OF [command]
    {
        [CR]               Connect_Request,
        [CC]               Connect_Confirm,
        [DR]               Disconnect_Request,
        [DC]               Disconnect_Confirm,
        [BC1]              Broadcast_Connect,
        [BC2]              Broadcast_Connect,
        [BC3]              Broadcast_Connect,
        [BR]               Broadcast_Repeat,
        [BS]               Broadcast_Stop,
        [BD]               Broadcast_Data
    }
}

```

-- coming from consumer
-- coming from producer
-- 3rd and 4th bits distinguish packet_kind
-- control message
-- data transfer
-- positive acknowledgement
-- negative acknowledgement
-- control
-- data transfer
-- positive acknowledgement
-- negative acknowledgement
-- last four bits of MTC octet indicate command
-- connect request
-- connect confirm
-- disconnect request
-- disconnect confirm
-- broadcast request first trial (optional)
-- broadcast request second trial (optional)
-- broadcast request third trial (optional)
-- broadcast stop (optional)
-- broadcast repeat (optional)
-- broadcast data (optional)
-- depending on MTC
-- see 2.3.7.5
-- see 2.3.7.5
-- see 2.3.7.5
-- see 2.3.7.5
-- see 2.3.7.5
-- see 2.3.7.5

Table 14 - Connect_Request

0	1	2	3	4	5	6	7
cl	0	0	0	0	0	0	0
cr_conn_ref							
cr_msg_size							
cr_credit				cr_pack_size			
cr_session_header							
cr_data							
(up to 14 octets)							

Connect_Request ::= RECORD

```

{
  cr_conn_ref          UNSIGNED16  -- 16-bit Connection_Reference
  cr_msg_size          UNSIGNED32  -- Total message size in octets,
                                   max 4GB
  cr_credit            UNSIGNED4   -- Proposed credit as:
                                   '0000'B = 0 (stop
                                   communication)
                                   '0111'B = 7 (maximum)
  cr_pack_size         ENUM4       -- Proposed packet size,
  {
    MVB_PACKET (0),              -- Transport_Data = 22
                                   octets/packet (MVB)
    WTB_PACKET (1)              -- Transport_Data = 123
                                   octets/packet (WTB)
                                   other values reserved.
  },
  cr_session_header    Am_Result   -- AM_OK in a call requesting a
                                   reply
  cr_data              ARRAY [14] OF WORD8 -- up to 14 octets of
                                   Transport_Data
}

```

Table 15 - Connect_Confirm

0	1	2	3	4	5	6	7
cl	1	0	0	0	0	0	1
cc_conn_ref							
cc_credit				cc_pack_size			


```

Connect_Confirm ::= RECORD
{
  cc_conn_ref          UNSIGNED16      -- same value as received in the
                                     Connect_Request.
  cc_credit            UNSIGNED4       -- Accepted credit, as:
                                     '0000'B = 0 (stop
                                     communication)
                                     '0111'B = 7 (maximum)
  cc_pack_size         ENUM4          -- Accepted packet size
  {
    MVB_PACKET (0),      -- Transport_Data_Size = 22
                          octets / packet (MVB)
    WTB_PACKET (1)      -- Transport_Data_Size = 123
                          octets / packet (WTB)
  }
  -- other values reserved.
}

```

Table 16 - Disconnect Request

0	1	2	3	4	5	6	7
cl	co	0	0	0	0	1	0
dr_reason							

```
Disconnect_Request ::= RECORD
{
  dr_reason          Am_Result  -- reason for disconnecting
}
```

Table 17 - Disconnect Confirm

0	1	2	3	4	5	6	7
cl	co	o	0	0	0	1	1

dc reason

[illegible]

Table 18 - Data_Packet

0	1	2	3	4	5	6	7
cl	0	0	1	dt_e	dt_seq_nr		
dt_data							
(up to Transport Data Size octets)							

Table 19 - Ack_Packet

0	1	2	3	4	5	6	7
cl	1	1	0	ak_e	ak_seq_nr		

```

Ack_Packet ::= RECORD
{
    ak_e          ENUM1 (=0)          -- always FALSE (0)
    ak_seq_nr     UNSIGNED3           -- number of next packet
                                         expected by Consumer.
}
-- last four bits of MTC octet

```

Table 19 - Ack Packet

0	1	2	3	4	5	6	7
cl	1	1	0	ak e	ak seq nr		

[illegible]

~~Table 20 - Nak Packet~~

	0	1	2	3	4	5	6	7
CL	1	1	1	nk_e	nk_seq_nr			

```
Nak_Packet ::= RECORD
{
  nk_eot                BOOLEAN1
                        -- last four bits of MTC octet
                        -- always FALSE (0)
  nk_seq_nr             UNSIGNED3
                        -- number of next packet
                        -- expected by Consumer
}
```

The following packets are used only in conjunction with Multicast (option):

Table 21 - Broadcast_Connect (BC1, BC2, BC3)

0	1	2	3	4	5	6	7
1	0	0	0	1	0	bc_rept	
bc_run_nr							
bc_msg_size							
bc_data							
(up to 18 octets)							

```

Broadcast_Connect ::= RECORD
{
    bc_run_nr            UNSIGNED16
    bc_msg_size          UNSIGNED16
    bc_data              ARRAY [18] OF WORD8
}
-- specified here for 2.3.7.5
-- BC1, BC2 and BC3 are
-- distinguished by bc_rept
-- Connection_Reference
-- identifying this message
-- Total message size in octets
-- up to 18 octets of user data

```

Table 22 - Broadcast_Data

0	1	2	3	4	5	6	7
1	0	0	0	1	1	0	0
bd_run_nr							
bd_offset							
bd_data							

```

Broadcast_Data ::= RECORD
{
    bd_run_nr            UNSIGNED16
    bd_offset            UNSIGNED16
    bd_data              ARRAY [18] OF WORD8
}
-- specified here for 2.3.7.5
-- Connection_Reference
-- identifying this message
-- offset of bd_data in octets
-- within message
-- up to 18 octets of user data

```

Table 23 - Broadcast_Repeat

0	1	2	3	4	5	6	7
1	1	0	0	1	1	0	1
br_run_nr							
br_offset							

```

Broadcast_Repeat ::= RECORD
{
    br_run_nr            UNSIGNED16
    br_offset            UNSIGNED16
}
-- specified here for 2.3.7.5
-- Connection_Reference
-- identifying this message
-- offset in octets of missed
-- bd_data

```

Table 24 - Broadcast_Stop (BSC, BSO)

0	1	2	3	4	5	6	7
1	co	0	0	1	1	1	0
br_run_nr							

```

Broadcast_Stop ::= RECORD                                -- specified here for 2.3.7.5
{
  bs_run_nr          UNSIGNED16                        -- Connection_Reference
                                                         identifying this message
}

```

2.3.6.6 State transition diagrams

2.3.6.6.1 States

The Message Transport Protocol shall implement for each Producer or Consumer an MTP state machine with the states listed in Table 25.

Table 25 - MTP states

State Name	Occurs at	Short Description
DISC	Producer, Consumer	disconnected
SETUP	Producer	waiting for CC packet
SEND	Producer	open for sending a message
SEND_CANC	Producer	DR packet sent, waiting for DC or DR packet
CLOSED	Producer	DR packet received
LISTEN	Consumer	waiting for acknowledgement of CC packet
RCV_CANC	Consumer	DR packet sent, waiting for DC or DR packet
RECEIVE	Consumer	open for receiving a message and all received packets acknowledged
PEND_ACK	Consumer	open for receiving a message and not all received packets acknowledged
FROZEN	Consumer	complete message with DT (EOT) packet received
LISTEN_FROZEN	Consumer	complete message with CR packet received

The states and transitions of a transport layer instance (Producer or Consumer) are illustrated in Figure 30 and specified in the following subclauses.

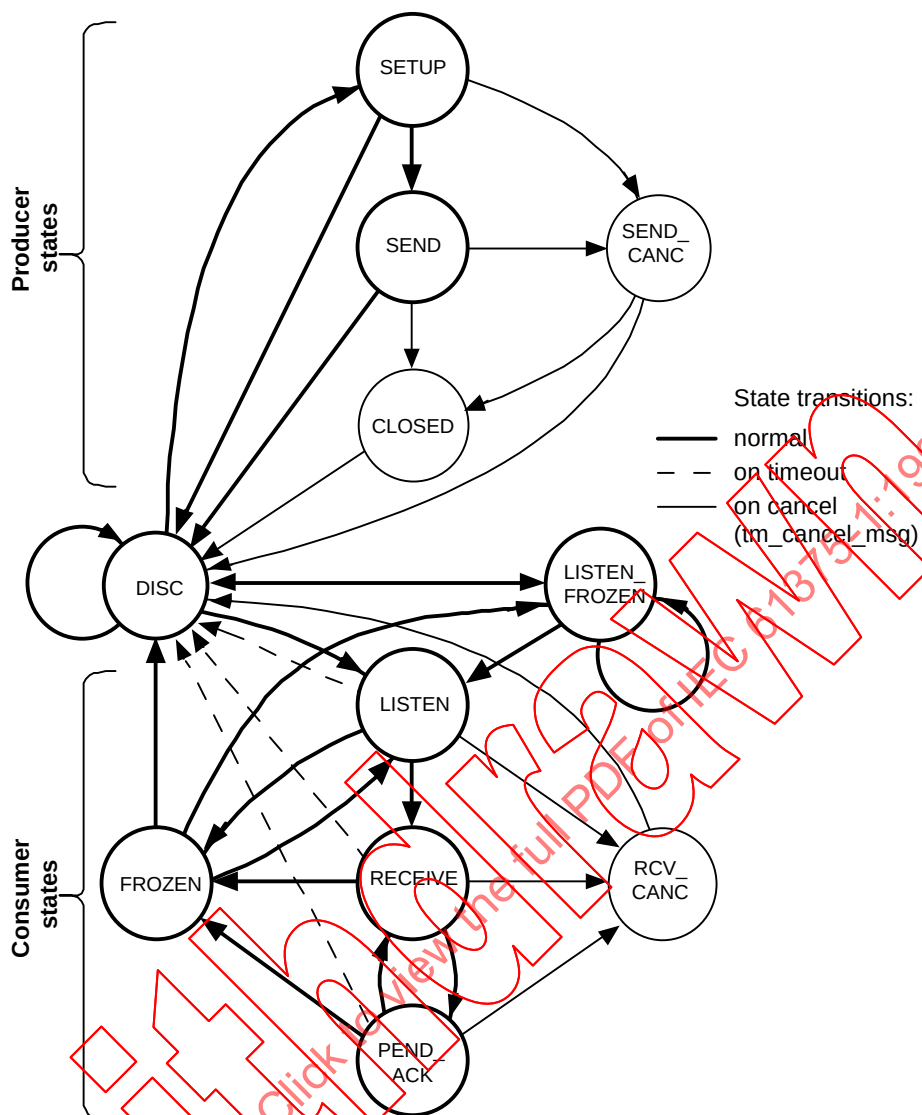


Figure 30 - State Transition diagram of the MTP

2.3.6.6.2 Incoming events

The incoming events listed in Table 26, generated by the network, the user or time-outs, may cause transitions from one MTP state to another.

Table 26 - MTP incoming events

Event Name	Interface	Short Event Description
tm_send_req	from user	request to send a message, the user passes a message buffer to the messenger
tm_cancel_req	from user	cancels an incoming or outgoing message transfer, the messenger returns the message buffer to the user if the cancel operation is successful
rcv_CR	from net	CR packet received
rcv_CC	from net	CC packet received
rcv_DT	from net	DT packet received
rcv_AK	from net	AK packet received
rcv_NK	from net	NK packet received
rcv_DR	from net	DR packet received
rcv_DC	from net	DC packet received
TMO	Internal	a time-out expired (there is only one running at a time)

2.3.6.6.3 Outgoing events

The outgoing events listed in Table 27, generated by the MTP state machine, may cause transitions from one MTP state to another in another MTP machine.

Table 27 - MTP outgoing events

Event Name	Interface	Short Event Description
tm_connect_ind	Indication to user	the user shall accept or reject the incoming Connect_Request; the user passes a message buffer to the messenger if the user accepts the message
tm_receive_ind	Indication to user	complete message received or receive error, the messenger returns the message buffer to the user
tm_send_cnf	Confirmation to user	complete message sent or send error, the messenger returns the message buffer to the user
send_CR	to net	transmission of a CR packet
send_CC	to net	transmission of a CC packet
send_DT	to net	transmission of a DT packet
send_AK	to net	transmission of a AK packet
send_NK	to net	transmission of a NK packet
send_DR	to net	transmission of a DR packet
send_DC	to net	transmission of a DC packet

2.3.6.6.4 Control parameters in the packets

The exchanged packets shall contain the control parameters listed in Table 28.

Table 28 - MTP control parameters

Event	Fields	Short Field Description
send_CR (	CR_conn_ref,	connection reference
rcv_CR (	CR_credit,	proposed credit
	CR_pack_size)	proposed packet size code
send_CC (	CC_conn_ref,	connection reference
rcv_CC (	CC_credit,	accepted credit
	CC_pack_size)	accepted packet size code
send_DT (	DT_seq_nr,	packet number
rcv_DT (	DT_eot)	end of message transfer flag
send_NK (	NK_seq_nr)	next expected packet number
rcv_NK (		
send_AK (	AK_seq_nr)	next expected packet number
rcv_AK (		
send_DR (	DR_reason)	reason for disconnection, Am_Result error code
rcv_DR (		

For send events (send_xx) actual parameters are specified for the control fields, whereas for receive events (rcv_xx) the formal field names are referenced in the actions.

2.3.6.6.5 Auxiliary variables

The state transition diagram relies the auxiliary variables listed in Table 29 to reduce the number of states. Some variables exist at the Producer only or at the Consumer only and belong to a connection, while global variables are used for all connections in common.

Table 29 - MTP auxiliary variables

Variable Name	Context	Short Variable Description
my_credit	Global	my maximum accepted credit
run_nr	Global	next free Connection_Reference
my_pack_size	Global	my maximum accepted packet size code
cancelled	Producer, Consumer	this message has been cancelled by the user
credit	Producer, Consumer	accepted credit for this connection
expected	Producer, Consumer	lower send (Producer) or receive (Consumer) edge
conn_ref	Producer, Consumer	Connection_Reference of this connection
rep_cnt	Producer, Consumer	count of repetitions
new_cnt	Consumer	count of unacknowledged packets
next_send	Producer	next packet number for transmission
send_not_yet	Producer	upper send window edge
eot	Producer	complete message transmitted
size	Producer	accepted packet size for this connection
error	Producer	AM_xxx error code for report with send confirm

2.3.6.6.6 Actions and time-outs

Each connection shall have its own timer for time-out supervision.

The timer shall be manipulated by the actions 'restart_tmo' and 'reset_tmo' and shall generate a TMO internal event when it expires.

The state transitions are governed by three time-outs:

- SEND_TMO, the send time-out at the Producer,
- ACK_TMO, the acknowledge time-out at the Consumer, and
- ALIVE_TMO, the alive time-out at the Consumer.

If the Producer receives no acknowledgement packet for a packet if sent within SEND_TMO, the Producer shall send the same packet again.

If the Producer receives no acknowledgement packet for a packet if sent after a number MAX_REP_CNT of unsuccessful attempts, the Producer shall disconnect.

The maximum value for the repetition count MAX_REP_CNT shall be three.

The Consumer shall acknowledge packets immediately when the credit is exhausted or when the message is completely received.

The maximal value for a credit AM_MAX_CREDIT shall be seven.

Otherwise, the Consumer shall delay the acknowledgement by the acknowledge time-out ACK_TMO, in order to acknowledge several packets with a single acknowledgement.

The value of the send time-out SEND_TMO is a configuration parameter.

The send time-out SEND_TMO shall be greater than twice the sum of the worst case transmission delay of the network (NET_DELAY), plus the packet processing time in a station (PROC_DELAY), plus the acknowledge time-out ACK_TMO as shown in Figure 39.

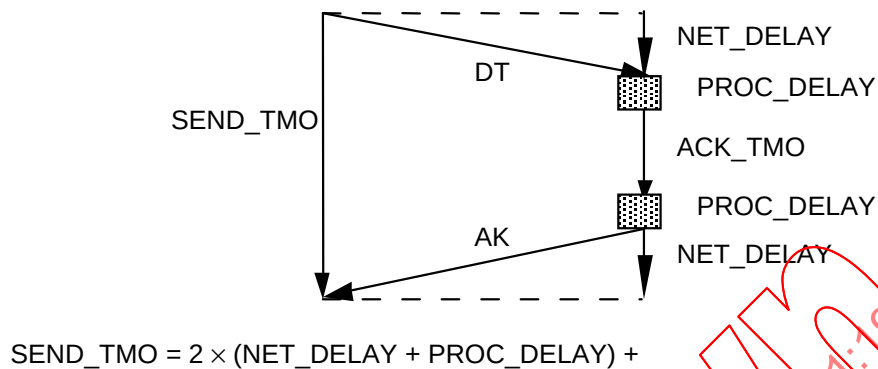


Figure 31 - Time-out SEND_TMO

The Consumer of a packet shall expect the next packet (with the same or a different sequence number) within ALIVE_TMO. If it receives no packets during this time, it shall disconnect the conversation.

The value of the alive time-out ALIVE_TMO is a configuration parameter.

The value of the alive time-out ALIVE_TMO shall be at least MAX_REP_CNT times the value of the send time-out SEND_TMO, plus the lifetime of the packets PACK_LIFE_TIME in the link queues as shown in Figure 40.

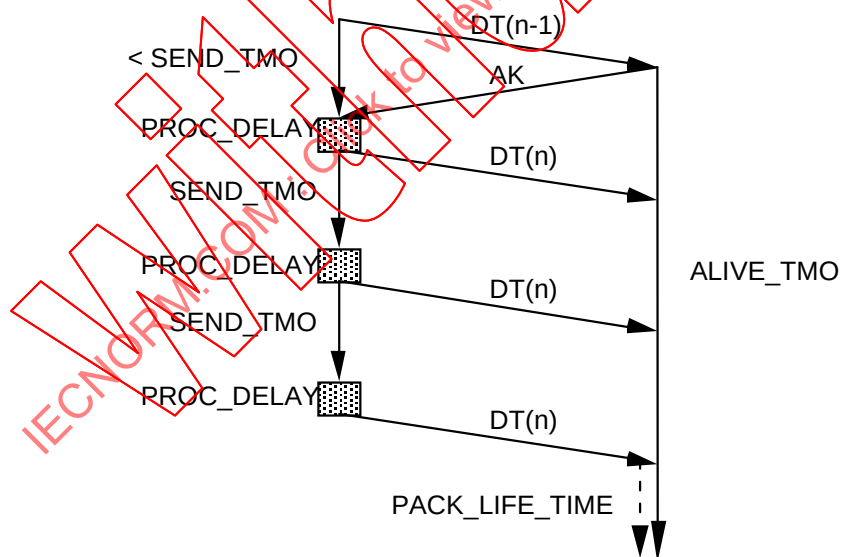


Figure 32 - Time-out ALIVE_TMO

The default values for a worst-case application are summarised in Table 30.

Table 30 - MTP time-outs (worst case)

	NET_ DELAY	PROC_ DELAY	PACK_LIFE_ TIME	ACK_ TMO	SEND_ TMO	ALIVE_ TMO
same bus	0,5 s	64 ms	5,0 s	0,25 s	1,4 s	9,5 s
vehicle bus to vehicle bus over train bus	9,6 s	64 ms	5,0 s	2,5 s	21,0 s	71,0 s

NOTE PACK_LIFE_TIME is specified in the corresponding link layer.

2.3.6.6.7 Implicit actions

Action descriptions in the following state transitions may contain the implicit actions referenced in Table 31.

Table 31 - Implicit actions

Reference	Implicit Action
(1)	AM_BUS_ERR status is signalled if no CR packet could be sent on the bus
(2)	operations with packet numbers modulo 8, comparisons regarding turnaround
(3)	cancelled is always TRUE in this case.

2.3.6.6.8 Compound Actions

The actions listed in Table 32 are executed in several state transitions.

Table 32 - Compound actions

[illegible]

2.3.6.6.9 Producer states event table

A Producer in a current state shall, upon occurrence of an event, transit to the next state and perform the action specified in Table 33.

Table 33 - Producer states and transitions

Current State	Event	Action(s)	Next state (if <> current)
(any)	tm_cancel_req	cancelled := TRUE;	
DISC	tm_send_req	update (eot); send_CR (run_nr, AM_MAX_CREDIT, my_pack_size); conn_ref := run_nr; run_nr := run_nr + 1; restart_tmo (SEND_TMO); expected := 0; rep_cnt := 0; cancelled := FALSE;	SETUP
SETUP	rcv_DR	close_send (DR_reason);	DISC
	rcv_CC AND (conn_ref = CC_conn_ref)	IF (eot) THEN close_send (AM_OK); ELSE credit := CC_credit; size := decode (CC_pack_size); next_send := 0; send_not_yet := credit; send_data_or_cancel; END;	DISC
	TMO AND (rep_cnt = MAX_REP_CNT)	close_send (AM_CONN_TMO_ERR); (1)	DISC
	TMO AND (rep_cnt < MAX_REP_CNT)	send_CR (conn_ref, AM_MAX_CREDIT, my_pack_size); rep_cnt := rep_cnt + 1; restart_tmo (SEND_TMO);	

Table 33 (continued)

Current State	Event	Action(s)	Next state (if <> current)
SEND	rcv_DR	send_DC; error := DR_reason; restart_tmo (ALIVE_TMO);	CLOSED
	rcv_AK AND (expected < AK_seq_nr <= send_not_yet) (2)	expected := AK_seq_nr; send_not_yet := expected + credit; (2) IF (NOT eot) THEN send_data_or_cancel; ELSIF (expected = next_send) THEN close_send (AM_OK); END;	SEND or SEND_CANC
	TMO AND (rep_cnt < MAX_REP_CNT)	local variable: seq_nr; seq_nr := expected; WHILE (seq_nr < (next_send-1)) DO (2) send_DT (seq_nr, FALSE); END; send_DT (next_send-1, eot); (2) rep_cnt := rep_cnt + 1; restart_tmo (SEND_TMO);	DISC
	TMO AND (rep_cnt = MAX_REP_CNT)	close_send (AM_SEND_TMO_ERR);	DISC
	rcv_NK AND (expected < NK_seq_nr <= send_not_yet) (2)	expected := NK_seq_nr; send_not_yet := expected + credit; (2) IF (NOT eot) THEN IF cancelled THEN send_DR (AM_REM_CANC_ERR); restart_tmo (SEND_TMO); ELSE REPEAT update (eot); send_DT (next_send, eot); (2) next_send := next_send + 1; UNTIL (eot OR (next_send = send_not_yet)); restart_tmo (SEND_TMO); END;	SEND_CANC

Table 33 (continued)

Current State	Event	Action(s)	Next state (if<>current)
SEND_CANC	rcv_DC	close_send; (3)	DISC
	rcv_DR	restart_tmo (ALIVE_TMO); (3)	CLOSED
	TMO AND (rep_cnt = MAX_REP_CNT)	close_send; (3)	DISC
	TMO AND (rep_cnt < MAX_REP_CNT)	send_DR (AM_REM_CANC_ERR); rep_cnt := rep_cnt + 1; restart_tmo (SEND_TMO);	
CLOSED	TMO	close_send (error);	DISC

2.3.6.6.10 Consumer states event table

A Consumer in a current state shall, upon occurrence of an event, transit to the next state and perform the action specified in Table 34.

Table 34 - Consumer states and transitions

Current State	Event	Action(s)	Next state (if <> current)
(any)	tm_cancel_req	cancelled := TRUE;	
DISC	rcv_DR	send_DC;	
DISC or FROZEN	rcv_CR	cancelled := FALSE; new_cnt := 0; expected := 0; tm_connect_ind (VAR err); IF (err = AM_OK) THEN conn_ref := CR_conn_ref; credit := min (my_credit, CR_credit); send_CC (conn_ref, credit, min (CR_pack_size, my_pack_size);	
LISTEN_FROZEN	rcv_CR AND (CR_conn_ref <> conn_ref)	IF (eot) THEN close_rcv (AM_OK); ELSE restart_tmo (ALIVE_TMO); END; ELSE send_DR (err); END;	LISTEN_FROZEN LISTEN DISC
LISTEN_FROZEN	rcv_CR AND (CR_conn_ref = conn_ref)	send_CC (conn_ref, credit, min (CR_pack_size, my_pack_size);	
FROZEN OR LISTEN_FROZEN	TMO		DISC

Table 34 (continued)

Current State	Event	Action(s)	Next state (if <> current)
LISTEN	rcv_CR AND (CR_conn_ref = conn_ref)	send_CC (conn_ref, credit, min (CR_pack_size, my_pack_size); restart_tmo (ALIVE_TMO);	
	rcv_CR AND (CR_conn_ref <> conn_ref)	close_rcv (AM_FAILURE);	DISC
	TMO	close_rcv (AM_ALIVE_TMO_ERR);	DISC
LISTEN or RECEIVE or PEND_ACK	rcv_DR	send_DC; close_rcv (DR_reason);	DISC
	rcv_DT AND cancelled	send_DR (AM_REM_CANC_ERR); rep_cnt := 0; restart_tmo (SEND_TMO);	RCV_CANC
	rcv_DT AND NOT cancelled AND (DT_seq_nr = expected)	expected := expected + 1; (2) new_cnt := new_cnt + 1; IF (DT_eom) THEN send_AK (expected); close_rcv (AM_OK); ELSIF (new_cnt = credit) THEN send_AK (expected); new_cnt := 0; restart_tmo (ALIVE_TMO);	FROZEN RECEIVE
		ELSIF (state = RECEIVE) OR (state = LISTEN) THEN restart_tmo (ACK_TMO); END;	PEND_ACK
	rcv_DT AND NOT cancelled AND (DT_seq_nr <> expected)	send_NK (expected); new_cnt := 0; restart_tmo (ALIVE_TMO);	RECEIVE
RECEIVE	TMO	close_rcv (AM_ALIVE_TMO_ERR);	DISC

Table 34 (concluded)

Current State	Event	Action(s)	Next state (if<>current)
PEND_ACK	TMO	send_AK (expected); new_cnt := 0; restart_tmo (ALIVE_TMO);	RECEIVE
FROZEN	rcv_DT	send_AK (expected);	
RCV_CANC	TMO AND (rep_cnt < MAX_REP_CNT)	send_DR (AM_REM_CANC_ERR); restart_tmo (SEND_TMO); rep_cnt := rep_cnt + 1;	
	TMO AND (rep_cnt = MAX_REP_CNT)	close_rcv (AM_FAILURE); (3)	DISC
	rcv_DR OR rcv_DC	close_rcv (AM_FAILURE); (3)	DISC

2.3.6.7 Transport Message Interface

The Transport_Message_Interface provides the services of the transport layer to the Session layer.

This interface may remain internal to a device. This interface is not covered by conformance testing.

The following specifications are included to improve portability.

2.3.6.7.1 Data exchange at the transport level

Figure 33 shows the interaction between transport layer and Session layer. The bullets indicate in which direction the parameters are passed.

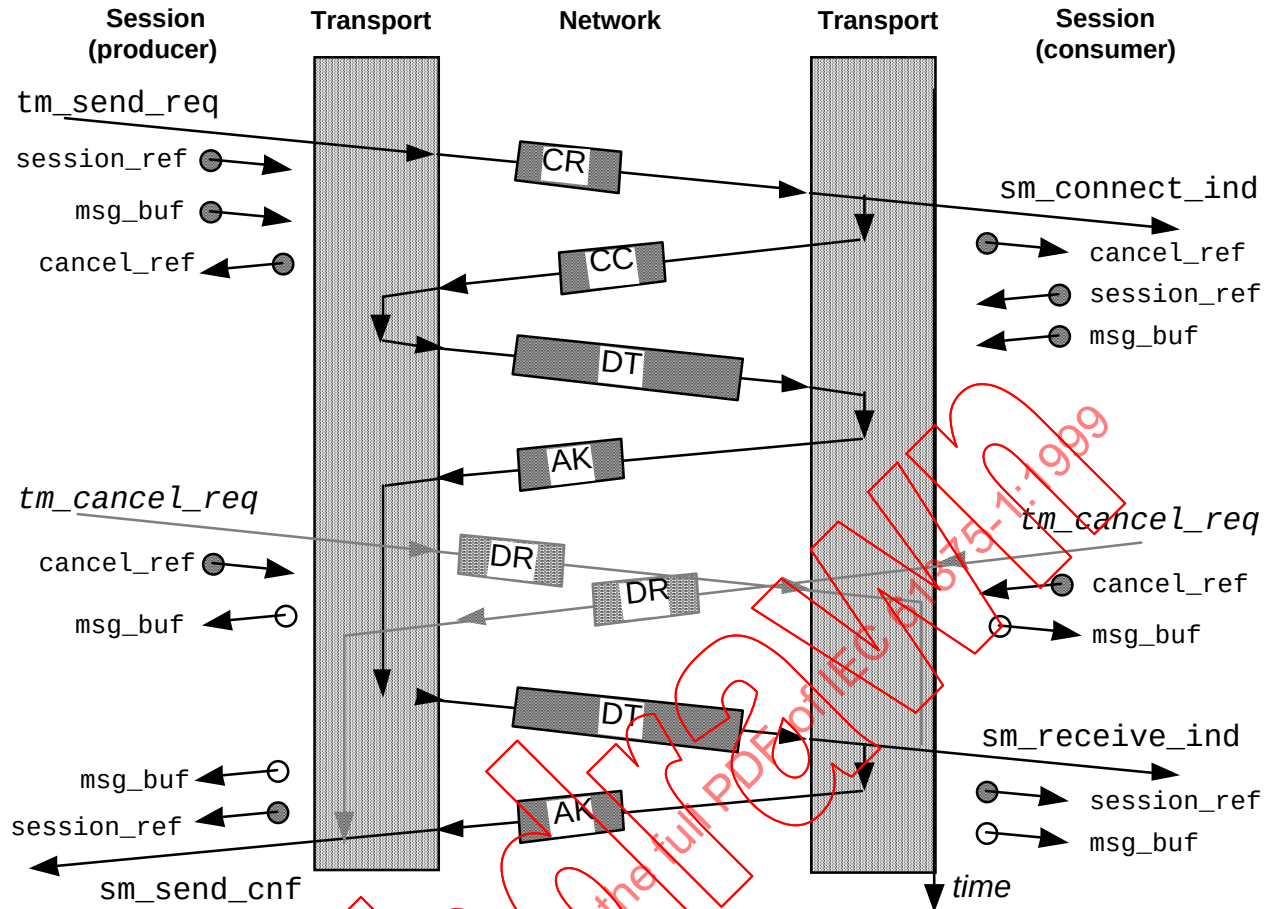


Figure 33 - Transport interface

The Producer sends a message contained in `msg_buf` by calling `tm_send_req`, identifying the connection through a session reference, 'session_ref'.

When the transfer is completed (successfully or not), the transport layer calls `sm_send_cnf`, (with the same session_ref), which then releases the message buffer.

At the Consumer site, the transport layer calls `sm_connect_ind` to notify the Session layer of the Connect_Request. If the Session layer accepts the Connect_Request, it supplies a buffer `msg_buf` for the transport layer to put the message.

The transport layer calls `sm_received_ind` when the message has been completely received (or was cancelled by the other party). This calls contains 'session_ref'.

Either party can cancel message transfer by calling `tm_cancel_req`.

The primitives of the TMI are prefixed by `tm_`.

The transport layer calls procedures of the Session layer (prefixed by `sm_`) which were previously subscribed and which have a type defined by the transport layer (prefixed by `TM_`).

The TMI is defined by the constants, types and procedures listed in Table 35 and specified in the following subclauses.

Table 35 - TMI primitives

Name	Meaning
Constants	
TM_CALLER_USER	user is a Caller
TM_REPLIER_USER	user is a Replier
Types	
TM_MSG_DESCR	message descriptor
TM_CONV_ID	conversation identifier
Initialisation Procedure	
tm_define_user	announces a user
Producer Procedures	
tm_send_req	sends a message
TM_SEND_CNF	confirms sending, indication procedure, of type of sm_send_cnf
sm_send_cnf	called when the message arrived (or was cancelled)
Consumer Procedures	
TM_CONNECT_IND	type of sm_connect_ind
sm_connect_ind	indicates arrival of a Connect_Request
TM_RECEIVE_IND	type of sm_receive_ind
sm_receive_ind	called when a message was completely received
Consumer and Producer Procedure	
tm_cancel_req	cancels the message

2.3.6.7.2 Type 'TM_MSG_DESCR'

Definition	Type of a message
Syntax	typedef struct {} TM_MSG_DESCR;
Usage	hidden structure used as a handle for cleaning up locked data structures.

2.3.6.7.3 Type 'TM_CONV_ID'

Definition	Type of a Conversation_Id
Syntax	<pre>typedef struct str_conv_id { UNSIGNED8 my_fct_or_station; UNSIGNED8 node; UNSIGNED8 function_or_station; UNSIGNED8 next_station } TM_CONV_ID;</pre>

2.3.6.7.4 Procedure 'tm_define_user'

Definition	Subscribes the indication procedures for a user identified as a Caller or as a Replier.	
Syntax	<pre>AM_RESULT tm_define_user (UNSIGNED who, TM_CONNECT_IND sm_connect_ind, TM_RECEIVE_IND sm_receive_ind, TM_SEND_CNF sm_send_cnf);</pre>	
Input	Who	user of the transport layer, either TM_CALLER_USER or TM_REPLIER_USER
	sm_connect_ind	procedure of the Consumer to call when a Connect_Request packet arrives. See TM_CONNECT_IND type definition.
	sm_receive_ind	procedure of the Consumer to call when a message has been completely received or has been cancelled by the Producer or upon error. See TM_RECEIVE_IND type definition.
	sm_send_cnf	procedure of the Producer to call when the message has been completely received by the Consumer or has been cancelled by the Consumer or on error. See TM_SEND_CNF type definition.
Return	any AM_RESULT	

2.3.6.7.5 Procedure 'tm_send_req'

Definition	Requests transfer of a message	
Syntax	<pre> AM_RESULT tm_send_req (UNSIGNED session_user, TM_CONV_ID * conversation, void * msg_addr, UNSIGNED32 msg_len, void * hdr_addr, UNSIGNED hdr_len, void * session_ref, TM_MSG_DESCR * * cancel_ref, UNSIGNED8 my_topo); </pre>	
Input	session_user	either TM_CALLER_USER (Call Message) or TM_REPLIER_USER (Reply Message)
	Conversation	Conversation_Id (concatenation of the Caller and the Replier address)
	msg_addr	pointer to the message body
	msg_len	total size of the message body
	hdr_addr	pointer to the Session_Header
	hdr_len	total size of the Session_Header
	session_ref	session reference linking 'tm_send_req' with the corresponding 'sm_send_cnf'.
	my_topo	value of the Topo_Counter supplied by the application, or 0 if unknown (= pass-all)
Return		any AM_RESULT
Output	cancel_ref	handle of the locked data structures of the transport layer.
Usage	1 - The buffer for the message to send and the buffer to place its Session_Header are separate. 2 - 'session_ref' is supplied by the transport layer user and is returned by the transport layer when calling 'sm_send_cnf'. 3 - 'cancel_ref' allows to release locked data structures. 'cancel_ref' is no longer valid after a successful cancel operation or after 'sm_send_cnf' returns.	

2.3.6.7.6 Type 'TM_SEND_CNF'

Definition	When a message has been completely received or cancelled by the Consumer, or to report an error, the transport layer shall call the Session layer procedure sm_send_cnf, which is of this type.	
Syntax	<pre>typedef void (* TM_SEND_CNF) (void * /* session_ref */ UNSIGNED8 /* my_topo */ AM_RESULT status);</pre>	
Input	session_ref	session reference which couples 'sm_send_cnf' with the previous 'tm_send_req'.
	my_topo	if result is AM_OK, Topo_Counter presently valid at the Node
	Status	AM_OK if the message has been acknowledged by the Consumer, otherwise an error code is given.
Usage	1 - For each tm_send_req, the transport layer calls the Session layer procedure sm_send_cnf, which is of the type TM_SEND_CNF, to inform the Producer that its message has been received or cancelled by the Consumer, or to report an error. 2 - 'sm_send_cnf' is not called if the connection has been successfully cancelled by the Producer. 3 - status is an input parameter.	

2.3.6.7.7 Type 'TM_CONNECT_IND'

Definition	When it receives a Connect_Request packet, the transport layer at the Consumer side shall call the procedure 'sm_connect_ind' of type TM_CONNECT_IND. If 'sm_connect_ind' accepts the message, it is expected to return a result AM_OK and supply a buffer for the message. If 'sm_connect_ind' returns a result different from AM_OK, the transport layer shall reject the connection and send a Disconnect_Request using as 'reason' parameter the result of 'sm_connect_ind'.	
Syntax	<pre>typedef void (* TM_CONNECT_IND) (const TM_CONV_ID*, /* in: conversation */ void * *, /* out: msg_addr */ UNSIGNED32, /* in: msg_len */ void * *, /* out hdr_addr */ UNSIGNED *, /* out: hdr_len */ TM_MSG_DESCR *, /* in: cancel_ref */ void * *, /* out: session_ref */ UNSIGNED8, /* in: my_topo */ AM_RESULT /* out: status */);</pre>	
Input	Conversation	Conversation_Id (concatenation of Caller and Replier address)
	msg_len	size of the message buffer the Session layer supplies
	cancel_ref	handle on locked data structures in the transport layer
	my_topo	Topo_Counter at the Node at the time the Connect_Request was received.
Output	msg_addr	pointer to the message body supplied by the Session layer.
	hdr_addr	pointer to the Session_Header of the message.
	hdr_len	total size of the Session_Header
	session_ref	supplied by the Session layer to identify the locked data structures in the Session layer. Links sm_connect_ind with the corresponding sm_receive_ind.
	Status	AM_OK if the Session layer accepts the connection, reject reason (AM_RESULT) otherwise.
Usage	The Session layer is expected to subscribe a procedure sm_connect_ind, of type TM_CONNECT_IND, in the tm_define_user procedure.	

2.3.6.7.8 Type 'TM_RECEIVE_IND'

Definition	When it completely received an incoming message, the transport layer shall call the procedure <code>sm_receive_ind</code> of type <code>TM_RECEIVE_IND</code> .	
Syntax	<pre>typedef void (* TM_RECEIVE_IND) (void *, /* session_ref */ AM_RESULT /* status */);</pre>	
Input	<code>session_ref</code>	reference which couples <code>sm_receive_ind</code> with the previous <code>sm_connect_ind</code> .
	Status	<code>AM_OK</code> when the message is successfully received and placed into the buffer, otherwise another <code>AM_RESULT</code> error code.
Usage	1 - The Session layer is expected to supply a procedure '<code>sm_receive_ind</code>' of type <code>TM_RECEIVE_IND</code>, in the '<code>tm_define_user</code>' procedure. 2 - '<code>sm_receive_ind</code>' is expected to return to the Session layer the buffers which the Session layer supplied in its '<code>sm_connect_ind</code>'	

2.3.6.7.9 Procedure 'tm_cancel_req'

Definition	Cancels a connection when called by either the Producer or the Consumer.	
Syntax	<pre>AM_RESULT tm_cancel_req (TM_MSG_DESCR * cancel_ref);</pre>	
Input	<code>cancel_ref</code>	handle on the locked data structures of the transport layer.
Return		<code>AM_OK</code> if cancel applied to an ongoing message, <code>AM_FAILURE</code> otherwise.
Usage	1 - If the result is <code>AM_OK</code>, the transport layer will have sent a <code>Disconnect_Request</code> packet and will not have '<code>sm_send_cnf</code>' or '<code>sm_receive_ind</code>' will not have been called. 2 - If the message has been already completely sent and acknowledged, this procedure has no effect. 3 - '<code>cancel_ref</code>' allows to release locked data structures, it is no more defined after a successful cancel operation.	

2.3.7 Multicast Transport Protocol (option)

2.3.7.1 Multicast packet exchange

The multicast transport shall be divided into three phases:

- a) connection establishment;
- b) acknowledged data transfer;
- c) disconnection.

A simple packet exchange with no packet losses is shown in Figure 34.

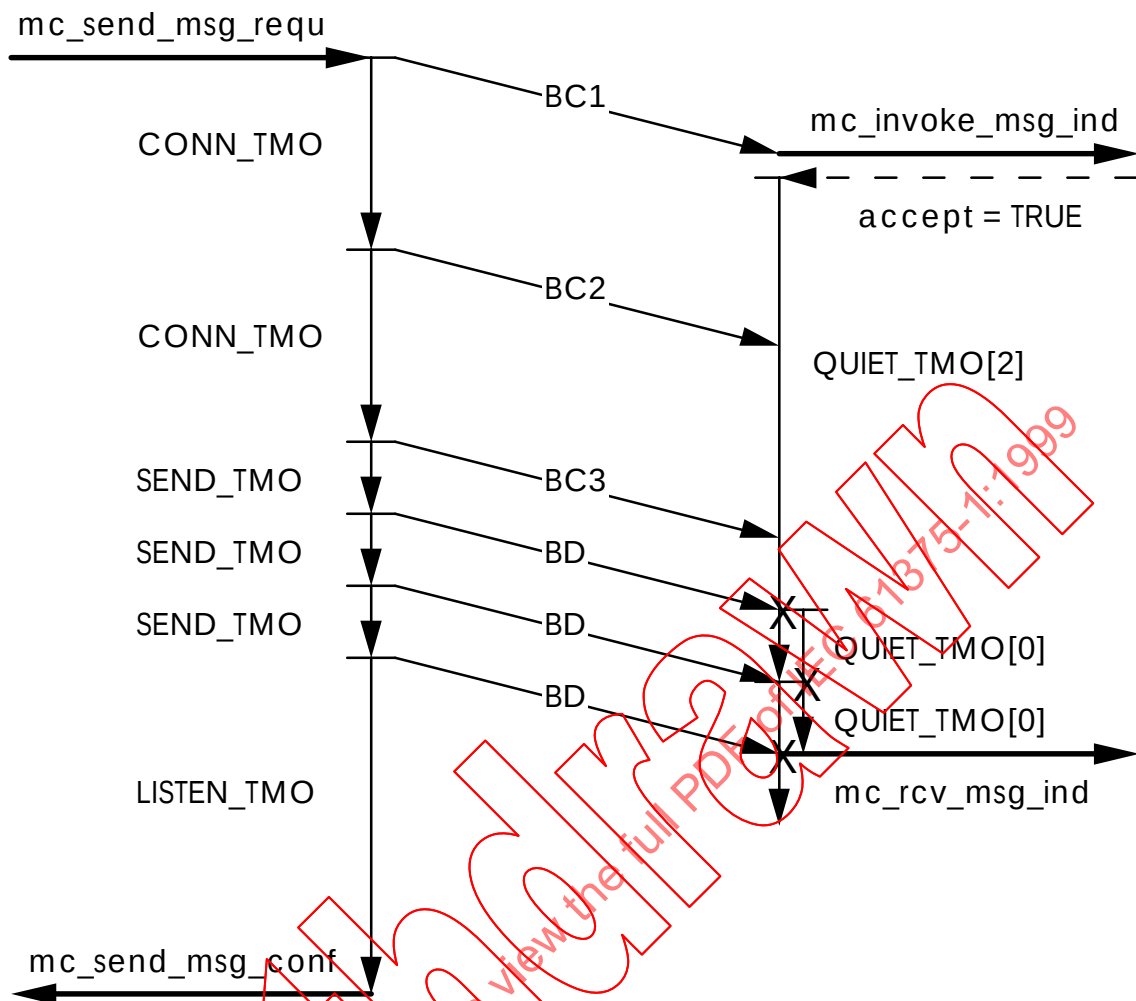


Figure 34 - Multicast message with no retransmission

2.3.7.1.1 Connection establishment

The first packet of a message is a Broadcast_Connect (BC) packet, which indicates the total size of the message and contains the first data segment.

When a BC packet arrives, the Consumer checks whether there is an application ready to receive the message. If there is none, the packet is discarded. Otherwise, the connection is opened, and the first data segment is copied into the application's message buffer.

The Producer always repeats the BC packet three times with a long delay (CONN_TMO) between retransmissions.

It is sufficient for the Consumers to get one of these three copies.

The BC packet also contains a retransmission countdown number, which allows the Consumers to adjust their time-out.

The Consumers cannot request repetition of BC.

If the message is short enough to fit within BC, the situation shown in Figure 35 occurs:

- the Producer Messenger confirms completion of transfer immediately after sending the last BC.
- the Consumer Messenger notifies the application and starts a time-out (ALIVE_TMO []) to suppress duplicates of the message due to the repeated BCs. This time-out depends on the BC's repetition count. Reception of the last BC stops the time-out and closes the connection.

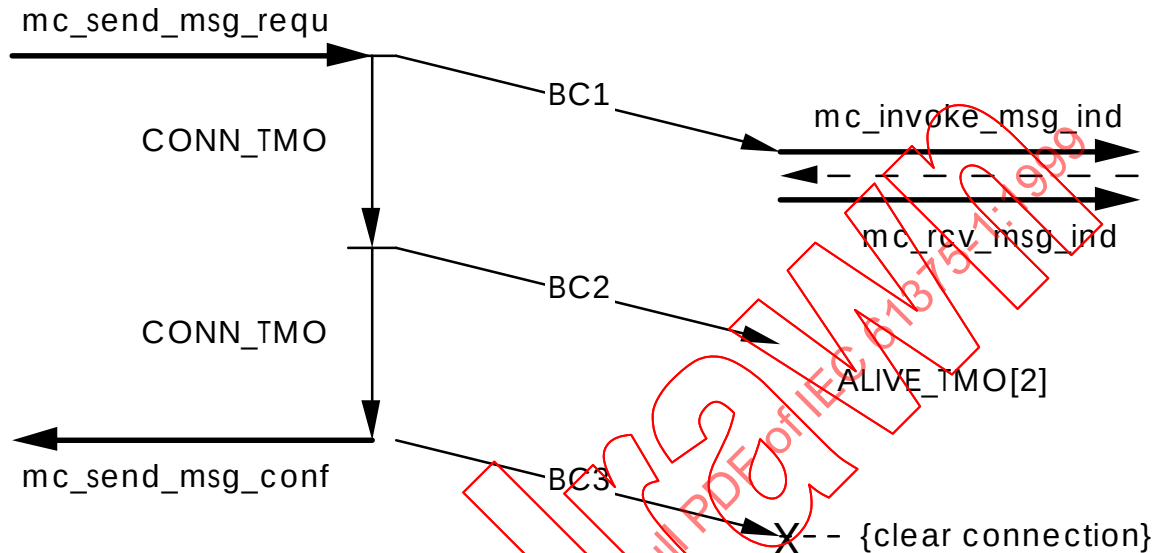


Figure 35 - Short multicast message with no BD packets and no loss

2.3.7.1.2 Data transfer

If the message is not yet complete with BC, the Producer starts transmitting Broadcast_Data (BD) packets at an interval SEND_TMO which is shorter than CONN_TMO.

The BD packets contain a 16-bit offset which indicates the position of the data segment within the whole message.

The Consumer which expects data starts a time-out QUIET_TMO to supervise the activity. If this time-out expires, or if the Consumer detects a packet with a higher offset than expected, the Consumer assumes that some BD packets have been lost and therefore it issues a Broadcast_Repeat (BR) packet to request retransmission. The BR packets contain a 16-bit offset from which retransmission is requested.

While sending BD packets, the Producer filters incoming BR packets and starts retransmission after insertion of a transmission pause (PAUSE_TMO in addition to the normal SEND_TMO).

With sending BR, the Consumer starts a REPEAT_TMO to supervise the effect of BR.

The Consumer may send a BR only if the REPEAT_TMO is not yet running at that moment, otherwise REPEAT_TMO might expire and cause a retransmission of BR, while on a second time-out the message would be discarded and the connection would be closed (three packets having been lost in this case).

Receiving the expected or an older BD restarts the time-out QUIET_TMO [0].

EXAMPLE - A situation in which packets went lost is shown in Figure 36.

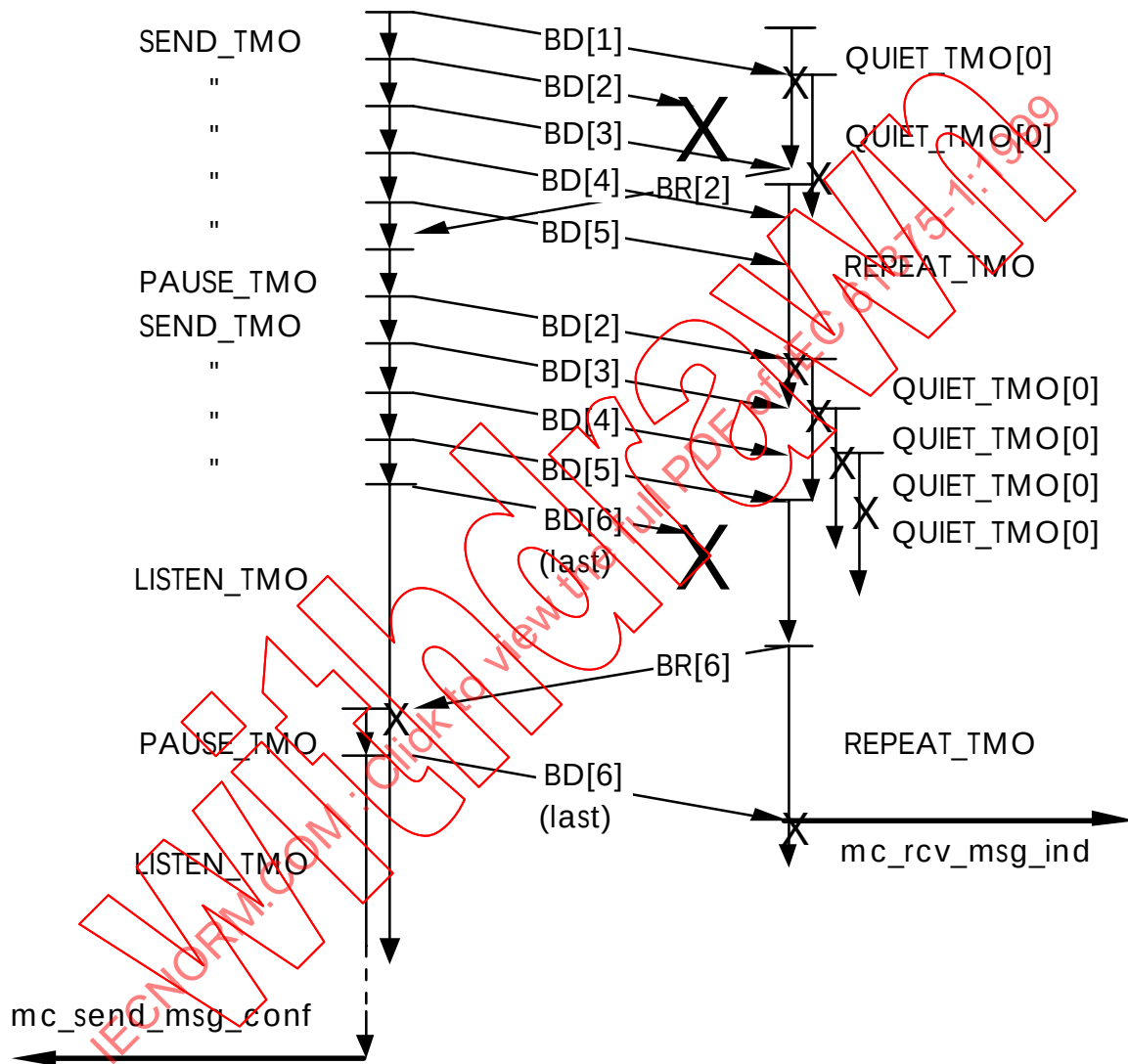


Figure 36 - Exchange with lost packets

2.3.7.1.3 Disconnection

After the last BD, the message is passed to the Consumer application and the connection is closed.

After the last BD packet, the Producer waits for Broadcast_Repeat (BR) packets during LISTEN_TMO before closing and confirming completion of transfer.

2.3.7.1.4 Cancelling a Transfer

If a Consumer cancels a request, then an incoming message is just discarded and nothing is visible on the bus. But if a Producer cancels a message multicasting, then it is worth to transmit a special packet (Broadcast_Stop, BS) to prohibit that all Consumers time out and generate a large amount of even repeated BR packets.

2.3.7.1.5 Retransmission Algorithm

The same BR packet can be issued by more than one Consumer, but these packets will arrive with different delays at the Producer. It is also possible that the arriving BR packets do not indicate an increasing missing offset (which is the case in a point-to-point transfer). The problem cannot be solved by inserting a long delay before starting retransmission, because the time-outs on both sides depend on each other.

Therefore a certain BR filter mechanism is necessary at the Producer. The used filter can be described as follows:

- after acceptance of a BR packet, another incoming BR packet with the same retransmission offset is discarded during a time-out SKIP_TMO.
- only a limited number REP_LIMIT of BR packets with different retransmission offsets is guaranteed to be accepted within a sliding offset window of a fixed size REP_RANGE.
- further BR packets that exceed this limit for any window may or may not be discarded.

Incoming BR packets are treated according to the following algorithm: the Producer maintains a registration table for the requested retransmissions for each message. This registration table consists of REP_LIMIT entries which are all empty when transfer is started. An occupied entry contains the retransmission offset (as a key) and its own timer which is loaded with SKIP_TMO when a corresponding BR is accepted. With an incoming BR the registration table is inspected to find out whether the requested offset for retransmission is already registered. If there is such an entry with the same offset, then the BR is discarded whenever the timer of this entry is still running, or else it is accepted. If the requested offset is not yet registered, then the BR is accepted and a new entry is inserted until the table becomes full. Once the table being full, acceptance of the BR depends on the existence of an entry that can be released for reuse. The entry with the lowest offset is overwritten if either the offset of the new BR or an offset of one of the other entries falls out of a window that has its lower edge at the lowest registered offset. Otherwise the BR is discarded because REP_LIMIT BR requests within the same window have already been accepted. Once the registration table is full, it remains full forever (until the connection is closed).

With this algorithm and with only one Consumer of the message, all BR packets which exceed the limit REP_LIMIT for any window are discarded (if there is more than one Consumer, then acceptance of such a BR is still possible).

2.3.7.1.6 Message consistency across Inaugurations

A Consumer of a MC message saves the Final_Node which is in the network header of an incoming BC packet. Once a receive connection is open, subsequent packets are only accepted if they are addressed to the same Final_Node, or else the connection is closed with an error. The routing node ensure that the Final_Node field in packets that are forwarded to the VB changes with each inauguration (the Final_Node field contains an inauguration counter). This prevents that packets originating from different Nodes are assembled into an invalid, mixed message (the address of one Node may be assigned to another Node by an inauguration, and both Nodes may use the same Connection_Reference by accident).

If the Producer of a MC message is located on a VB end-device, he is not aware of an intermediate inauguration and continues sending BD packets until the complete message is transmitted.

2.3.7.2 Connection Identification

For each incoming packet, the connection to which the packet belongs is retrieved. At the receiving site, a connection is identified by a 32-bit structure composed of the own Final_Function, Origin_Function, Origin_Node and Origin_Station.

At the transmitting site, a connection is identified by the two involved functions only, because incoming BR packets may contain any origin node (there are several Consumers).

Messages with the same connection identification are transmitted in sequence.

2.3.7.3 Connection reference

Each packet contains a 16-bit Connection_Reference as message identification, which is assigned by the Producer incrementally to each outgoing message and initialised at random.

2.3.7.4 Multicast Message Transport Control encoding

The Multicast protocol uses the same network layer as the MTP protocol, thus the packets have the same Link_Header and Network_Header. The two protocols are distinguished by the Message Transport Control, which is specified in 2.3.6.5.2, 'Message Transport Control Encoding'

2.3.7.5 Multicast Packets encoding

The Multicast Protocol distinguishes the following packets as shown in Figure 37.

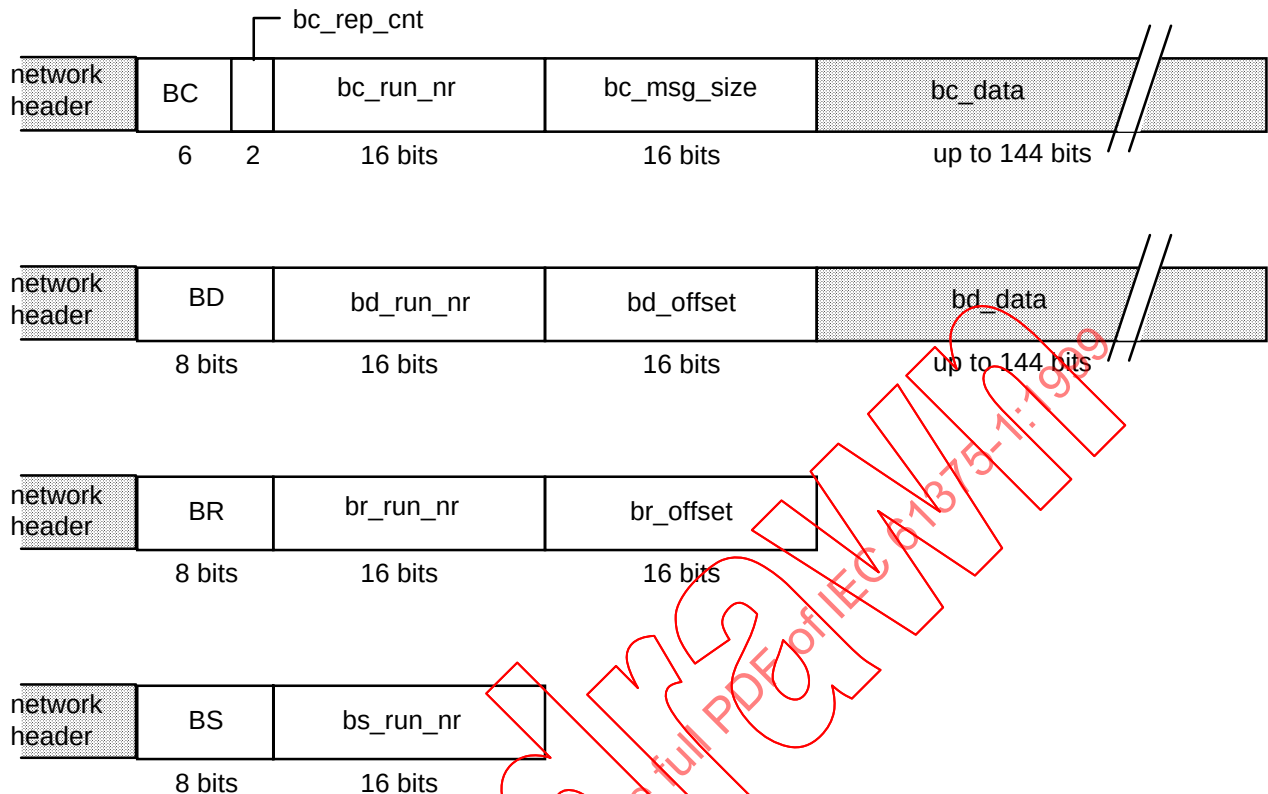


Figure 37 - Packet formats

Multicast Packets shall be encoded as specified in 2.3.6.5.4.

2.3.7.6 Multicast transport protocol definition

This multicast protocol operates over a connectionless, unreliable network which maintains the sequence of the packets and limits the lifetime of the packets to PACK_LIFE_TIME.

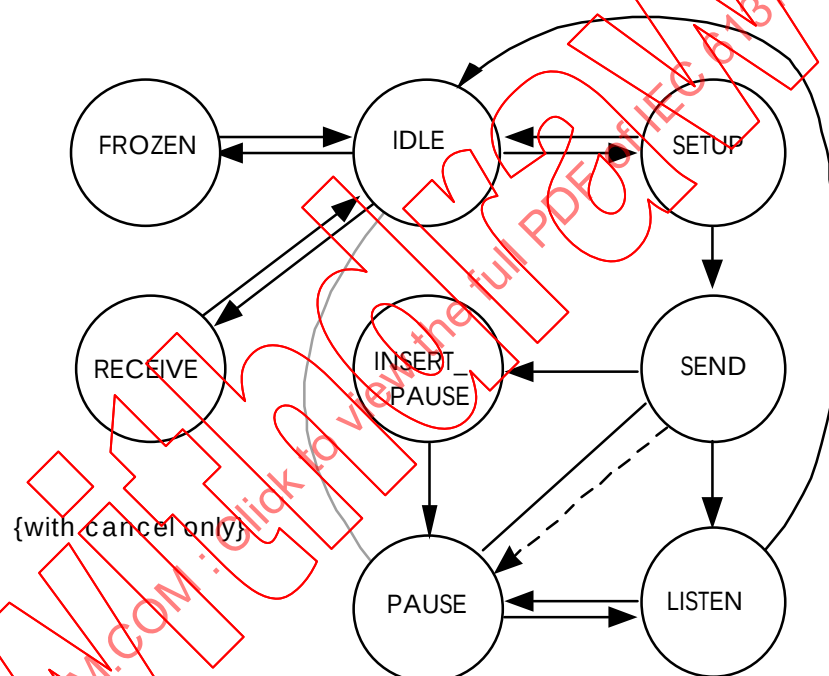
A connection is established for each message transfer. The Producer of the message is called Producer, whereas a receiving partner is called Consumer. One timer is necessary for each connection at the Consumer, while the Producer needs an additional timer for each registration table entry to implement SKIP_TMO.

2.3.7.6.1 States and state transition diagram

The multicast protocol machine, as illustrated in Figure 38, shall have the states listed in Table 36.

Table 36 - States of the MCP machine

State Name	Occurs at	Short State Description
IDLE	Producer, Consumer	disconnected
SETUP	Producer	sending BC packets
SEND	Producer	sending BD packets periodically
LISTEN	Producer	complete message sent, waiting for BR packets
INSERT_PAUSE	Producer	additional delay requested before sending next BD packet
PAUSE	Producer	additional delay is running
RECEIVE	Consumer	BC packet received and BD packets expected
FROZEN	Consumer	some but not last BC packet received and no BD packets expected

**Figure 38 - Protocol machine states**

The transitions between these states depend on incoming events defined in Table 37 and produce outgoing events defined in Table 38

Table 37 - Incoming Events

Event Name	Occurs at	Interface	Short Event Description
mc_send_msg_requ	Producer	from user	request to send a message, the user passes a message buffer to the Messenger
mc_cancel_msg	Producer, Consumer	from user	cancels an incoming or outgoing message transfer, the Messenger returns the message buffer to the user if the cancel operation is successful
rcv_BC	Consumer	from net	BC packet received
rcv_BD	Consumer	from net	BD packet received
rcv_BS	Consumer	from net	BS packet received
rcv_BR	Producer	from net	BR packet received
skipped (reg_entry)	Producer	internal	SKIP_TMO expired for registration table entry 'reg_entry'
TMO	Producer, Consumer	internal	any other time-out expired

Table 38 - Outgoing Events

Event Name	Occurs at	Interface	Short Event Description
mc_invoke_msg_ind	Consumer	to user	incoming Connect_Request to be accepted or rejected by the user, the user passes a message buffer to the Messenger if the user accepts the message
mc_rcv_msg_ind	Consumer	to user	complete message received or receive error, the Messenger returns the message buffer to the user
mc_send_msg_conf	Producer	to user	complete message sent or send error, the Messenger returns the message buffer to the user
send_BC	Producer	to net	transmission of a BC packet
send_BD	Producer	to net	transmission of a BD packet
send_BS	Producer	to net	transmission of a BS packet
send_BR	Consumer	to net	transmission of a BR packet

2.3.7.6.2 Control fields in the data packet

The exchanged packets contain the control fields described in Table 39.

Table 39 - Control fields in packets

Event Name	Field Names	Short Field Description
xxx_BC (	BC_run_nr, BC_rep_cnt, BC_msg_size)	Connection_Reference count of pending BC retransmissions total message size
xxx_BD (	BD_run_nr, BD_offset)	Connection_Reference offset of data segment within the message
xxx_BS (	BS_run_nr)	Connection_Reference
xxx_BR (	BR_run_nr, BR_offset)	Connection_Reference missing offset

NOTE The abbreviation xxx stands for 'send' or 'rcv'. For xxx = send events actual parameters are specified for the control fields, whereas for xxx = rcv events the formal field names are referenced in the actions.

2.3.7.6.3 Auxiliary variables

The state transition diagram relies on auxiliary variables to reduce the number of states. Some variables exist at the Producer or Consumer only and belong to a connection, while other variables are global which means that they are used for all connections at a site in common. The auxiliary variables listed in Table 40 shall be implemented.

Table 40 - Auxiliary variables

Variable Name	Context	Short Variable Description
run_nr	global	next free Connection_Reference
cancelled	Producer, Consumer	this message has been cancelled by the user
timer	Producer, Consumer	timer for all time-outs except SKIP_TMO
msg_size	Producer, Consumer	total message size
offset	Producer, Consumer	offset of next data segment
conn_run_nr	Producer, Consumer	Connection_Reference of this connection
rep_cnt	Producer, Consumer	count of repetitions (BR or BC)
my_node	Consumer	Final_Node for this connection
reg_table[REP_LIMIT]. offset	Producer	registration table, retransmission offsets
reg_table[REP_LIMIT]. skip_timer	Producer	registration table, SKIP_TMO time-outs
reg_table[REP_LIMIT]. timer_running	Producer	registration table, boolean: 'skip_timer' is running.
table_is_full	Producer	boolean: registration table is full.
window_is_full	Producer	boolean: all registered offsets are within same window of size REP_RANGE.
next_entry	Producer	next free entry (index) in registration table, or entry with the lowest offset if 'table_is_full'

NOTE A timer can only be started or stopped (not rescheduled) and it generates an event on time-out.

2.3.7.6.4 Constants

Table 41 lists the constants used in the Multicast protocol machine.

Table 41 - MCP constants

Constant Name	Constant Value	Short Constant Description
REP_LIMIT	6	size of registration table (number of entries)
MAX_TRIALS	3	number of trials (transmitted BC packets), loss of one less packets is tolerated
BD_DATA_SIZE	18 (for MVB frame)	number of data octets in a BD packet
PROC_DELAY	1 x 64 [ms]	processing time for events (polling period)
NET_DELAY	8 x 64 [ms]	expected transmission delay (end-to-end)

REP_RANGE	LISTEN_TMO / SEND_TMO x BD_DATA_SIZE

NOTE Offset range within which retransmission is possible defines the window size REP_RANGE.
sdasdsadadsadsa

The time-outs shall be as specified in Table 42.

Table 42 - MCP time-outs

Time-out Name	Time-out Value
Producer site	
CONN_TMO	4 x 64 [ms]
SEND_TMO	1 x 64 [ms]
PAUSE_TMO	1 x 64 [ms]
SEND_MAX_TMO	SEND_TMO + PROC_DELAY + PAUSE_TMO
SKIP_TMO	REPEAT_TMO - NET_DELAY - PROC_DELAY
LISTEN_TMO	(QUIET_TMO[0] + PROC_DELAY) + (MAX_TRIALS - 2) x (REPEAT_TMO + PROC_DELAY) + 2 x NET_DELAY + PROC_DELAY - SEND_TMO
Consumer site	
QUIET_TMO [MAX_TRIALS]	SEND_MAX_TMO + PROC_DELAY + NET_DELAY + ix x (CONN_TMO + PROC_DELAY)
ALIVE_TMO [MAX_TRIALS]	NET_DELAY + PACK_LIFE_TIME + ix x (CONN_TMO + PROC_DELAY)
REPEAT_TMO	SEND_MAX_TMO + 2 x NET_DELAY + PROC_DELAY

NOTE 1 All time-outs and other time indications are specified in units of milliseconds. For those time-outs that can be chosen freely, a selected number is specified, while for dependent time-outs a formula for the minimum value is given.

NOTE 2 The notation 'ix' means 'array index 0 .. (MAX_TRIALS - 1)'

2.3.7.6.5 Compound Actions

The compound actions listed in Table 43 are used several times in the MCP machine.

Each connection has its own timer for time-out supervision. The timer is manipulated by the actions 'restart_tmo' and 'reset_tmo' and generates an internal event 'TMO' on expiring. At the Producer, an additional timer is used for each SKIP_TMO which is manipulated through 'restart_skip' or 'reset_skip' and it generates an internal event 'skipped (reg_entry)' on expiring. All timers that belong to a connection are reset with disconnection.

Table 43 - MCP Compound actions

Action Name	Short Action Description	Next State
restart_tmo (xxx_TMO)	stops the 'timer' and starts it with time-out xxx_TMO;	
reset_tmo	stops the 'timer';	
restart_skip (reg_entry)	starts 'skip_timer' with SKIP_TMO and sets 'timer_running' = TRUE for 'reg_table[reg_entry]';	
reset_skip (reg_entry)	stops 'skip_timer' and sets 'timer_running' = FALSE for 'reg_table[reg_entry]';	
close_send	IF (NOT cancelled) THEN mc_send_msg_conf; END; reset_tmo; FOR ix := 0 TO (REP_LIMIT - 1) DO reset_skip (ix); END;	IDLE
close_rcv (error)	IF (NOT cancelled) THEN mc_rcv_msg_ind (error); END; reset_tmo;	IDLE
accept_BC	cancelled := FALSE; my_node := final_node; offset := 0; mc_invoke_msg_ind (VAR err); IF (err = AM_OK) THEN conn_run_nr := BC_run_nr; msg_size := BC_msg_size; update (offset); IF (offset = msg_size) THEN IF (NOT cancelled) THEN mc_rcv_msg_ind (AM_OK); END; IF (BC_rep_cnt = 0) THEN ELSE restart_tmo (ALIVE_TMO [BC_rep_cnt]); END; ELSE rep_cnt := 0; restart_tmo (QUIET_TMO [BC_rep_cnt]); END; END; END;	IDLE FROZEN RECEIVE

The filtering of BR packets is done by check_BR, which returns a boolean to signal whether BR is accepted or not. All auxiliary variables that appear only at the Producer are accessed by check_BR exclusively except for initialisation which is done by init_BR_filter. The filtering is specified in Table 44.

Table 44 - Filtering of BR packets

Action Name	Action Specification
init_BR_filter	<pre> table_is_full := FALSE; window_is_full := FALSE; next_entry := 0; FOR ix := 0 TO (REP_LIMIT - 1) DO reg_table[ix].timer_running := FALSE; END;</pre>
check_BR (VAR accept)	<pre> temporary variables: ix, max_offset; accept := FALSE; IF ((BR_run_nr <> conn_run_nr) OR (BR_offset >= offset)) THEN RETURN; END; IF (any entry ix registered with reg_table[ix].offset = BR_offset) THEN IF (NOT (reg_table[ix].timer_running)) THEN accept := TRUE; END; ELSIF (NOT (table_is_full)) THEN ix := next_entry; INC (next_entry); IF (next_entry = REP_LIMIT) THEN table_is_full := TRUE; END; accept := TRUE; ELSIF ((BR_offset - reg_table[next_entry].offset >= REP_RANGE) OR (NOT (window_is_full))) THEN ix := next_entry; accept := TRUE; END; IF (accept) THEN offset := BR_offset; restart_skip (ix); IF (table_is_full) THEN (* update next_entry as entry with lowest registered offset; *) (* update window_is_full; *) next_entry := 0; max_offset := reg_table[0].offset; FOR ix := 1 TO (REP_LIMIT-1) DO IF (reg_table[ix].offset < reg_table[next_entry].offset) THEN next_entry := ix; ELSIF (reg_table[ix].offset > max_offset) THEN max_offset := reg_table[ix].offset; END; END; window_is_full := (max_offset - reg_table[next_entry].offset) < REP_RANGE; END; END;</pre>

2.3.7.6.6 Producer state event table

Table 45 specifies the behaviour of the Producer.

Table 45 - MCP Producer state event table

Current State	Event	Action(s)	Next state
(any)	mc_cancel_msg	cancelled := TRUE;	
	skipped (reg_entry)	reg_table[reg_entry].timer_running := FALSE;	
IDLE	mc_send_msg_requ	conn_run_nr := run_nr; INC (run_nr); cancelled := FALSE; offset := 0; msg_size := requ_msg_size; rep_cnt := MAX_TRIALS - 1; init_BR_filter; send_BC (conn_run_nr, rep_cnt, msg_size); update (offset); restart_tmo (CONN_TMO);	SETUP
SETUP	TMO	IF (cancelled) THEN send_BS (conn_run_nr); close_send; ELSE DEC (rep_cnt); send_BC (conn_run_nr, rep_cnt, msg_size); IF (rep_cnt > 0) THEN restart_tmo (CONN_TMO); ELSIF (offset = msg_size) THEN close_send; ELSE restart_tmo (SEND_TMO); END; END;	IDLE SEND
SEND	rcv_BR	check_BR (VAR accept); IF (accept) THEN END;	INSERT_PAUSE
	TMO AND cancelled	restart_tmo (PAUSE_TMO);	PAUSE

Table 45 (concluded)

Current State	Event	Action(s)	Next state
SEND OR PAUSE	TMO AND NOT cancelled	send_BD (conn_run_nr, offset); update (offset); IF (offset = msg_size) THEN restart_tmo (LISTEN_TMO); ELSE restart_tmo (SEND_TMO); END;	LISTEN SEND
PAUSE	TMO AND cancelled	send_BS (conn_run_nr); close_send;	IDLE
PAUSE OR INSERT_PAUSE	rcv_BR	check_BR (VAR accept);	
INSERT_PAUSE	TMO	restart_tmo (PAUSE_TMO);	PAUSE
LISTEN	TMO	close_send;	IDLE
	rcv_BR	check_BR (VAR accept); IF (accept) THEN restart_tmo (PAUSE_TMO); END;	PAUSE

2.3.7.6.7 Consumer state event table

A Consumer in a current state shall, upon occurrence of an event, transit to the next state and perform the action specified in Table 46.

Table 46 (concluded)

Current State	Event	Action(s)	Next state
RECEIVE (continued)	(rcv_BS OR rcv_BD) AND ((BD_run_nr <> conn_run_nr) OR (my_node <> final_node))	close_rcv (AM_FAILURE);	IDLE
	rcv_BC AND ((BC_run_nr <> conn_run_nr) OR (my_node <> final_node))	close_rcv (AM_FAILURE); accept_BC;	IDLE IDLE OR FROZEN OR RECEIVE
	rcv_BS AND (BC_run_nr = conn_run_nr) AND (my_node = final_node)	close_rcv (AM_REM_CANC_ERR);	IDLE
FROZEN	TMO OR rcv_BS	reset_tmo;	IDLE
	rcv_BC AND (BC_run_nr = conn_run_nr) AND (my_node = final_node)	IF (BC_rep_cnt = 0) THEN reset_tmo; END;	IDLE
	rcv_BC AND ((BC_run_nr <> conn_run_nr) OR (my_node <> final_node))	reset_tmo; accept_BC;	IDLE IDLE OR FROZEN OR RECEIVE

2.3.8 Message session layer

2.3.8.1 Purpose

The Session layer pairs two messages: a Call_Message and a Reply_Message.

The Call_Message is sent by the Caller to the Replier, the Reply_Message is sent from the Replier to the Caller.

This pair of messages allows to implement a Remote Procedure Call as shown in Figure 39.

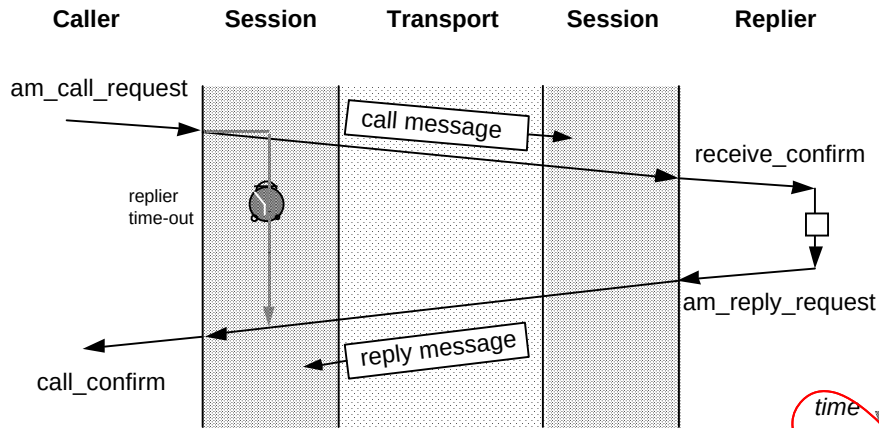


Figure 39 - Session layer transfer

The Session layer uses the services of the transport layer for each message transfer.

2.3.8.2 Conversation identifier

The Session layer shall uniquely identify the communicating partners through a Conversation_Id, which consists of the concatenation of:

- the Network_Address of the remote application,
- the Function_Id of the local application.

The Session layer shall reject a request for communication using a given Conversation_Id as long as another session with the same Conversation_Id is in progress.

The Session layer shall keep the full address of the Caller to forward the corresponding Reply_Message.

NOTE The Conversation_Id contains all information necessary to forward the packet to the network layer and to identify packets when they come from the network layer.

2.3.8.3 Shortcut

The Session layer shall shortcuts the network when the partner resides on the same station, i.e. forward the message directly without involving the transport layer.

2.3.8.4 Topo Counter check

The Session layer shall check the consistency between my_topo (supplied by the Application) and this_topo (value of the Topo_Counter kept by the network layer) according to the following algorithm:

```

IF (my_topo = AM_ANY_TOPO) THEN my_topo = this_topo ELSE
IF (this_topo = AM_ANY_TOPO) THEN this_topo = my_topo ELSE
IF (my_topo <> this_topo) THEN reject the call.
    
```

The Session layer shall use the value of the Topo_Counter for the duration of this conversation.

NOTE This ensures that a conversation is cancelled if an inauguration takes place between the Call and the Reply_Message.

2.3.8.5 Session header encoding

In the Connect_Request of a Call_Message, the Session_Header consists of one octet filled with '0'. All other combinations are reserved as shown in Figure 40.

In the Connect_Request of a Reply_Message, the Session_Header consists of one octet containing the reply status supplied by the Replier application.

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

Figure 40 - Session_Header in Call_Message (of type Am_Result)

2.3.8.6 Buffer management

The Session layer shall provide two buffer management:

- a) static buffers, which are allocated and deallocated by the Application and
- b) dynamic buffers, which are allocated and deallocated by the Session.

2.3.8.7 Session layer interface

The interface of the Session layer is identical with that of the Application layer since the presentation layer has no protocols.

2.3.9 Message Presentation Layer

The presentation layer for messages has no protocols.

Messages shall be transmitted as ARRAY OF WORD8, in ascending memory addresses.

The message headers and parameters shall observe the same data presentation rules as for Process_Variables (for instance, all data are transmitted most significant octet first).

The data types which shall be used in the protocols and which are recommended for the application are listed in 2.4.

2.3.10 Message Application layer

2.3.10.1 Purpose

The Application_Messages_Interface (AMI) allows an Application to send and receive messages over the network. The AMI offers a Call/Reply service, as well as initialisation and buffer management and a Multicast service.

The AMI is defined as a set of procedures which access the Session layer directly (the presentation layer and the Application layer have no protocols).

2.3.10.2 Application_Messages_Interface

2.3.10.2.1 AMI primitives

The Application Layer Interface shall implement the primitives illustrated in Figure 41, listed in Table 47 and specified in the following subclauses.

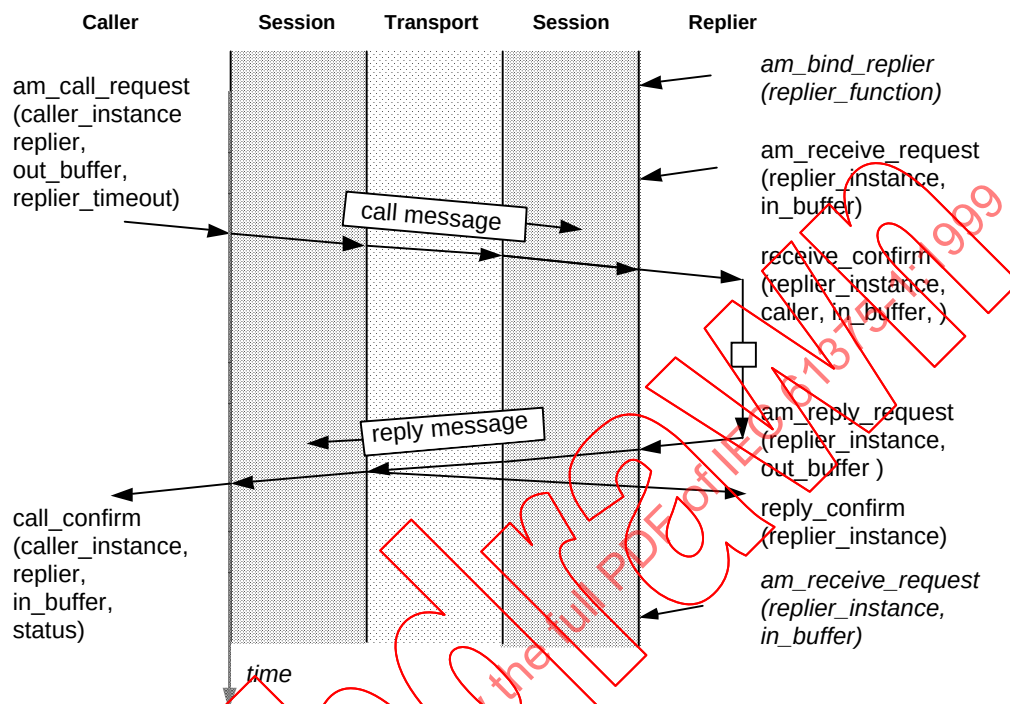


Figure 41 - Application_Messages_Interface

NOTE 1 AMI objects are prefixed with `am_` or `AM_` (for Application Messages), the objects belonging to the Caller or Replier instance are not prefixed.

NOTE 2 The following abbreviations are used in the names:

- REM - remote, failure reported by the partner device;
- LOC - local, failure reported by the own device;
- OVE - overflow;
- TMO - time-out.

Table 47 - AMI primitives

Name	Meaning
Constants and Types	
AM_RESULT	result of a procedure, same definition as Am_Result.
AM_ADDRESS	Network Address of remote entity
Initialisation	
am_init	initialises the Messenger
am_announce_device	configures a device
am_show_busses	lists the Bus_Id of the attached link layers
am_set_current_tc	indicates to the Messenger the current Topo_Counter
Station Directory interface	
AM_STADI_ENTRY	station directory entry
am_stadi_write	writes the station directory
am_stadi_read	reads the station directory
Function directory interface	
AM_DIR_ENTRY	function directory entry
am_clear_dir	initialises the function directory
am_insert_dir_entries	records the station identifier of a list of functions
am_remove_dir_entries	removes a list of functions
am_get_dir_entry	retrieves the station identifier of a given function
Group directory interface	
AM_GROUP	definition of a group
am_clear_groups	clears the group directory
am_insert_member	includes a Node into a group directory
am_remove_member	removes a Node from a group directory
am_member	membership in group directory
Caller interface	
am_call_request	Caller sends a whole message
AM_CALL_CONFIRM	type of the procedure called when the reply arrives
am_call_cancel	cancel the conversation and discard Reply_Message
Replier interface	
am_bind_replier	announce a Replier instance to the Session layer
am_unbind_replier	cancel the above announcement
am_receive_request	announce instance is ready for the next call
AM_RECEIVE_CONFIRM	type of the procedure called when call is complete
am_reply_request	called by the Replier instance to send back a Reply_Message
AM_REPLY_CONFIRM	type of the procedure called when reply sent
am_receive_cancel	cancels a ready or engaged Replier instance
Buffer handling	
am_buffer_free	recycles a dynamic message buffer

NOTE Interface procedures are non-blocking and task scheduling is not constrained.

2.3.10.2.2 Definition of AM_RESULT

Definition	A procedure which returns a value of type AP_RESULT shall encode it as follows:
Syntax	<pre> typedef enum { AM_OK =0, /* successful termination AM_FAILURE =1, /* unspecified failure AM_BUS_ERR =2, /* no bus transmission possible AM_REM_CONN_OVF =3, /* too many incoming connections AM_CONN_TMO_ERR =4, /* Connect_Request not answered AM_SEND_TMO_ERR =5, /* send time-out (connect was OK) AM_REPLY_TMO_ERR =6, /* no reply received AM_ALIVE_TMO_ERR =7, /* no complete message received AM_NO_LOC_MEM_ERR =8, /* not enough memory or timers AM_NO_REM_MEM_ERR =9, /* no more memory or timer (partner) AM_REM_CANC_ERR =10, /* cancelled by partner AM_ALREADY_USED =11, /* same operation already done AM_ADDR_FMT_ERR =12, /* address format error AM_NO_REPLY_EXP_ERR =13, /* no such reply expected AM_NR_OF_CALLS_OVF =14, /* too many calls requested AM_REPLY_LEN_OVF =15, /* Reply_Message too long AM_DUPL_LINK_ERR =16, /* duplicated conversation error AM_MY_DEV_UNKNOWN_ERR =17, /* my device unknown or not valid AM_NO_READY_INST_ERR =18, /* no ready Replier instance AM_NR_OF_INST_OVF =19, /* too many Replier instances AM_CALL_LEN_OVF =20, /* Call_Message too long AM_UNKNOWN_DEST_ERR =21, /* partner device unknown AM_INAUG_ERR =22, /* train inauguration occurred AM_TRY_LATER_ERR =23, /* (internally used only) AM_FIN_NOT_REG_ERR =24, /* final not registered AM_GW_FIN_NOT_REG_ERR =25, /* final not registered in router AM_GW_ORI_REG_ERR =26, /* origin not registered in router AM_MAX_ERR =31 /* highest system code. /* user codes are higher than 31 } AM_RESULT; </pre>

If an AMI procedure returns an application-dependent user code as result, it shall be greater than AM_MAX_ERR and less than 256.

NOTE AM_RESULT uses the same encoding as the Am_Result field transmitted in the packets.

2.3.10.2.3 Address constants

The constants listed in Table 48 are reserved identifiers.

Table 48 - Address constants

Constant	Code	Meaning
AM_SAME_STATION	0	this station, regardless of Station_Id
AM_UNKNOWN	255	unknown Station_Id
AM_MAX_BUSSES	1..16	maximum number of link layers supported by an implementation.
AM_ROUTER_FCT	251	Function_Id of Router
AM_AGENT_FCT	253	Function_Id of Agent
AM_MANAGER_FCT	254	Function_Id of Manager
AM_SAME_NODE	0	communication under the same Node, not going over the train bus.
AM_SYSTEM_ADDR	128	bit 0 of the Node_Address indicates a System_Address.
AM_ANY_TOPO	0	Topo_Counter is unknown.

2.3.10.2.4 Type 'AM_ADDRESS'

A Caller or a Replier shall identify the other party by its application address, which is of type AM_ADDRESS.

Definition	Type of an Application address (Caller or Replier).	
Syntax	<pre>typedef struct AM_ADDRESS - big-endian representation. { unsigned snu :1 /* bit 0 unsigned gni :1 /* bit 1 unsigned node_or_group :6 unsigned func_or_stat :8 unsigned next_station :8 unsigned topo_rsv :1 /* bit 0 unsigned topo_valid :1 /* bit 1 unsigned topo_counter :6 } AM_ADDRESS;</pre>	
Elements	snu	(system, not user) bit 0 = 0 indicates a User_Address bit 0 = 1 indicates a System_Address.
	gni	(group, not individual) bit 1 = 0, indicates an individual (Node) address, bit 1 = 1, indicates a Group address.
	node_or_group	Gni = 0, bits 2..7 specify a Node_Address gni = 1, bits 2..7 specify a Group_Address
	func_or_stat	if bit 0 of snu = 0, this is a Function_Id if bit 0 of snu = 1, this is a Station_Id
	next_station	Next_Station_Id
	res	reserved, always 0
	tpv	(topo valid) indicates if the following topo_counter is valid
	topo_counter	Topo_Counter or AM_ANY_TOPO.

NOTE The meaning of the fields differ at the Caller and at the Replier site as specified in the following.

A convenient encoding of an application address is shown in Figure 42:

0	1	2	3	4	5	6	7				
snu	gni	node_or_group									
func_or_stat											
next_station											
trv	tpv	topo_counter									

Figure 42 - Encoding of AM_ADDRESS.

NOTE AM_ADDRESS is an interface format, Am_Address is transmission format.

2.3.10.3 Caller side

2.3.10.3.1 Own identification

A Caller shall identify itself by its Function_Id.

NOTE The Caller identifies itself in 'am_call_request'

2.3.10.3.2 Caller Instances

As a Caller can set up several calls before receiving a reply, the variable 'caller_ref' shall link the 'am_call_request' with the corresponding 'call_confirm'.

2.3.10.3.3 System or User (snu)

If any other function than the Manager makes a call with a System_Address, the call shall not be executed and an address error shall be reported in 'call_confirm'.

NOTE Any function may call an Agent function or a Manager function through its User_Address, by specifying next_station, but only if the communication does not transit via the train bus (node = AM_SAME_NODE).

2.3.10.3.4 Group or Individual (gni)

If the Caller sets the 'gni' bit to '0', the single cast protocol shall be used and the following six bits shall be interpreted as a Node_Address.

If the Caller sets the 'gni' bit to '1', the multicast protocol shall be used and the following six bits shall be interpreted as a Group_Address.

NOTE The Multicast Protocol uses the same address format.

2.3.10.3.5 Node or Group (node_or_group)

If the Caller specifies (Node_Address <> AM_SAME_NODE), the call shall be forwarded to the train bus Node.

NOTE Even if the Replier's Node_Address is identical to the Caller's Node_Address, the message will be forwarded to the Node, which checks the Topo_Counter and reflect the message back over the vehicle bus.

2.3.10.3.6 Station or function (func_or_stat)

Any Function_Id may be used with a User_Address.

The Manager may specify (Station_Id = AM_UNKNOWN) in a System_Address, however, the call shall not be executed and an address error shall be reported in 'call_confirm' if 'next_station' is 'AM_UNKNOWN'.

NOTE 1 This allows a Manager to access a station which has an unknown Station_Id at initialisation time.

NOTE 2 When a call is sent over the train bus, Station_Id = 0 or 255 addresses the remote node, regardless of its Station_Id.

2.3.10.3.7 Next_Station

Next_Station specifies the Link_Address to which the Message shall be forwarded next. Next_Station can also specify the final station or a router station. It shall be computed as follows:

- a) If Next_Station is specified (Next_Station_Id $\neq$ AM_UNKNOWN), the Link_Address shall be taken from the station directory, using Next_Station_Id as entry.
- b) If the Next_Station is not specified (Next_Station_Id = AM_UNKNOWN), the Link_Address shall be taken from the station directory, using the following default as entry:
 - If the message is sent to the train bus (Node_Address $\neq$ AM_SAME_NODE) or (multicast), Next_Station_Id shall be taken from the function directory using the router function (AM_ROUTER_FCT) as entry.
 - If the message is not sent over the train bus (Node_Address = AM_SAME_NODE),
 - in case of a System_Address: Next_Station_Id shall be set equal to Station_Id;
 - in case of a User_Address: Next_Station_Id shall be retrieved from the function directory using Function_Id as entry.
- c) If (Next_Station_Id = AM_SAME_STATION) or (Next_Station_Id = this_station), the Messenger shall forward the Call_Message to a local Replier if it exists.

An address error shall be raised if the station directory has no entry for the Link_Address corresponding to Next_Station_Id.

NOTE If the Caller resides on a Node, next_station will be AM_SAME_STATION.

2.3.10.3.8 Topo_Counter

Bit 0 (most significant) of this octet shall be 0.

Bit 1 shall be 1 if the next six bits contain a valid Topo_Counter.

Otherwise, all bits of this octet shall be 0 (AM_ANY_TOPO).

An address error shall be raised if the application specifies the value AM_ANY_TOPO for any calls for which (node $\neq$ AM_SAME_NODE).

NOTE A Caller may not send a point-to-point message over the train bus if it ignores the topography. It is expected to first fetch the Topography from the Node or from an intermediate application. In case of a fixed train bus configuration, any value of Topo_Counter is acceptable.

2.3.10.3.9 Use of Network_Address at the Caller

The System_Address and User_Address modes are summarised in Table 49.

Table 49 - System Address and User Address

System_Address		same Node	other Node
Next_Station =	Station_Id =	Link_Address =	Link_Address =
AM_SAME_STATION	AM_SAME_STATION	shortcut to own station	Node_Address (error if this_station is not a Node)
	<> AM_SAME_STATION and <> AM_UNKNOWN	error	
	AM_UNKNOWN	shortcut to own station	
<> AM_SAME_STATION and <>AM_UNKNOWN	AM_SAME_STATION	stadi (next_station)	stadi (next_station) (Agent is on remote station)
	<> AM_SAME_STATION and <> AM_UNKNOWN		
	AM_UNKNOWN		
AM_UNKNOWN	AM_SAME_STATION	shortcut to own station	stadi (fundu(AM_ROUTER_FCT)) (Agent on remote Node)
	<> AM_SAME_STATION and <> AM_UNKNOWN	stadi (Station_Id)	stadi(fundu(AM_ROUTER_FCT)) (Agent on remote station)
	AM_UNKNOWN	error	stadi(fundu(AM_ROUTER_FCT)) (Agent on remote Node)
User_Address		same Node	other Node
Next_Station	Function_Id	Link_Address =	Link_Address =
AM_SAME_STATION	any	shortcut to own station	train bus, Node_Address (error if station is no Node)
<> AM_SAME_STATION and <> AM_UNKNOWN	any	stadi (next_station)	stadi(fundu(AM_ROUTER_FCT))
AM_UNKNOWN	any	stadi (fundu(Function_Id)) (error if function not registered)	stadi(fundu(AM_ROUTER_FCT)) or (train bus, Node_Address)

2.3.10.4 Replier site

2.3.10.4.1 Replier instances

Replier processes are Application Processes. Several Replier instances may service the same function in parallel. The caller may not specify which instance serves the call.

Each Replier function shall be bound before this function can call the 'am_receive_request' procedure to receive an incoming call and the 'am_reply_request' procedure to reply a received call.

The Replier processes are not blocked when waiting for a Call_Message or during transfer of a Reply_Message, instead they are notified when a Call_Message has been received or when transfer of the Reply_Message has completed.

The confirmation procedures to be called for notification are specified with the binding and are thus the same for all instances of the same Replier function.

A Replier instance shall reply or cancel each received call before it can issue a further 'am_receive_request'. Each request which is not yet confirmed can also be cancelled. A request that was successfully cancelled will not be confirmed.

2.3.10.4.2 Replier Identification

As a function can be executed by several instances, the variable 'replier_ref' shall link 'am_receive_request' with the corresponding receive_confirm, 'am_reply_request' and 'reply_confirm'.

A Replier instance shall be identified by its Function_Id and by its External_Reference.

NOTE the Session layer keeps the full address of the Replier as received in the Call_Message for the Reply_Message. The Session layer keeps the External_Reference as part of the Conversation_Id.

2.3.10.4.3 System or User (snu)

The 'snu' bit shall be '1' if the message has been received with a System_Address and in this case, the Agent function shall be called, the Caller being implicitly the Manager.

The 'snu' bit shall be set to '0' if the message has been received with a User_Address.

NOTE The Agent can be addressed by any other function over the User_Address, but only from stations attached to the same Node.

2.3.10.4.4 Group or Individual (gni)

The 'gni' bit shall be set to 1 to indicates that the message has been received over a multicast address, and set to 0 if it has been received over a single cast address.

NOTE This allows to call a Replier indifferently by the single-cast or by the multicast protocol.

2.3.10.4.5 Node or Group

Whether an individual or a Group_Address is used, the next six bits shall indicate the Node_Address of the Caller, or AM_SAME_NODE if the Caller specified AM_SAME_NODE in its Replier address.

NOTE If the Caller specifies the Node_Address, this address is delivered to the Replier even if the message does not travel over the train bus.

2.3.10.4.6 Next_Station

Next_Station shall be the Station_Id of the station over which the call has been received or, if the final station is the Node itself, Next_Station shall be AM_SAME_STATION.

2.3.10.4.7 Topo_Counter

The Replier shall receive the Topo_Counter of the Node to which it is attached if the message has been forwarded over the Node, otherwise, this field is set to AM_ANY_TOPO.

NOTE The Replier is responsible for checking that the value of the Topo_Counter of a Call_Message matches the 'my_topo' value.

2.3.10.5 Initialisation

The Messages Service is initialised at different level by the following procedures.

2.3.10.5.1 Procedure 'am_init'

Definition	Initialises the Messenger and calls am_clear_dir to initialise the directory.	
Syntax	AM_RESULT am_init (void);	
Result	AM_RESULT	AM_OK, AM_FAILURE
Usage	This procedure shall be called at system initialisation before calling any other am_xxx procedure.	

2.3.10.5.2 Procedure 'am_announce_device'

Definition	Announces the device's configuration.	
Syntax	<pre> AM_RESULT am_announce_device (UNSIGNED16 max_call_number, UNSIGNED16 max_inst_number, UNSIGNED16 default_reply_timeout, UNSIGNED8 my_credit); </pre>	
Input	max_call_number	number of simultaneous calls on this device.
	max_inst_number	number of simultaneous instances for any Replier on this device (default is three).
	default_reply_tmo	default reply time-out for call requests.
	my_credit	maximal (accepted) credit for all connections ending on this device and is cut to AM_MAX_CREDIT.
Return		any AM_RESULT
Usage	This procedure obtains the Node_Address directly from the link layer.	

2.3.10.5.3 Procedure 'am_show_busses'

Definition	Retrieves the number of link layers (busses) connected to this station and lists their Bus_Id.	
Syntax	AM_RESULT	am_show_busses (UNSIGNED8 * nr_of_busses, UNSIGNED8 link_id_list [AM_MAX_BUSSES]);
Return		any AM_RESULT
Output	nr_of_busses	number of connected link layers (also the number of elements in link_id_list)
	link_id_list	list of the link layers, with at least the Bus_Id of each.

2.3.10.5.4 Procedure 'am_set_current_tc'

Definition	Sets the current value of the Topo_Counter 'this_topo' for the network layer.	
Syntax	AM_RESULT UNSIGNED8	am_set_current_tc (this_topo);
Input	this_topo	Topo_Counter or AM_ANY_TOPO if the Topo_Counter is unknown.
Return		AM_OK if (AM_ANY_TOPO ≤ this_topo < 63), AM_FAILURE otherwise.
Usage	1 - an Application (Caller or Replier) is expected to obtain the current value of the Topo_Counter and copy it to its 'my_topo' variable. This value differs normally from Application to Application, even within the same Node, since an inauguration can take place at any time and applications are not expected to be notified of every change. 2 - The way through which the application received the Topo_Counter is not specified: it can be through management messages, through direct access to the link layer on a Node, through a periodic variable, etc. 3 - If this_topo is not equal to AM_ANY_TOPO, the Messenger will reject subsequent calls made with a Topo_Counter value not equal to this one or not equal to AM_ANY_TOPO, with a result AM_INAUG_ERR in 'call_confirm'. 4 - The value of this_topo is initially AM_ANY_TOPO.	

2.3.10.6 Station directory interface

The station directory is optional. Simple systems can do the mapping in a fixed way. In case a station directory is used, it is made available through the following procedures.

2.3.10.6.1 Type 'AM_STADI_ENTRY'

Definition	Type of a station directory entry	
Syntax	<pre>typedef struct { UNSIGNED8 station; UNSIGNED8 next_station; ENUM8 bus_id; UNSIGNED64 device_adr; } AM_STADI_ENTRY;</pre>	
Elements	station	Station_Id (key to retrieve next_station)
	next_station	Next_Station For a directly reachable station, Next_Station is equal to Station_Id
	bus_id	Bus_Id
	device_adr	Device_Address (bus-dependent)

2.3.10.6.2 Procedure 'am_stadi_write'

Definition	Inserts a number of entries into the station directory, each entry is checked for validity and consistency and may be rejected.	
Syntax	<pre>AM_RESULT am_stadi_write (const AM_STADI_ENTRY entries[], UNSIGNED8 nr_of_entries);</pre>	
Input	entries	list of new station directory entries.
	nr_of_entries	number of elements in entries.
Return	AM_OK: all entries could be successfully written into the station directory. AM_FAILURE: any entry in the list did not pass the checks. In this case, only these entries are rejected.	

2.3.10.6.3 Procedure 'am_stadi_read'

Definition	Reads a number of entries from the station directory.	
Syntax	<pre>AM_RESULT am_stadi_read (AM_STADI_ENTRY entries[], UNSIGNED8 nr_of_entries);</pre>	
Input	entries[].station	entries to be read.
	nr_of_entries	number of entries.
Return	AM_OK, AM_FAILURE	
Output	entries[].next_station	the fields entries[].next_station is the output information.

2.3.10.7 Function directory interface

2.3.10.7.1 Type 'AM_DIR_ENTRY'

Definition	Type of a function directory entry.
Syntax	<pre>typedef struct AM_DIR_ENTRY { UNSIGNED8 function; UNSIGNED8 station; } AM_DIR_ENTRY;</pre>

2.3.10.7.2 Procedure 'am_clear_dir'

Definition	Sets the Station_Id of all functions in the directory to AM_UNKNOWN.
Syntax	<pre>AM_RESULT am_clear_dir (void);</pre>
Output	AM_OK, AM_FAILURE

2.3.10.7.3 Procedure 'am_insert_dir_entries'

Definition	Inserts the Station_Id for each Function_Id listed in the first 'number_of_entries' elements of the list function_list.	
Syntax	<pre>AM_RESULT am_insert_dir_entries (AM_DIR_ENTRY * function_list, unsigned number_of_entries);</pre>	
Input	function_list	function directory list
	number_of_entries	number of elements in this list.
Return		AM_OK, AM_FAILURE

2.3.10.7.4 Procedure 'am_remove_dir_entries'

Definition	Sets the Station_Id to AM_UNKNOWN for each Function_Id listed in the first 'number_of_entries' elements of the list function_list	
Syntax	<pre>AM_RESULT am_remove_dir_entries (AM_DIR_ENTRY * function_list, unsigned number_of_entries);</pre>	
Input	function_list	function directory list
	number_of_entries	number of elements in this list.
Return		AM_OK, AM_FAILURE

2.3.10.7.5 Procedure 'am_get_dir_entry'

Definition	Reads the Station_Id for a given Function_Id from the function directory.	
Syntax	<pre> AM_RESULT am_get_dir_entry (UNSIGNED8 function, UNSIGNED8 *station); </pre>	
Input	function	Function_Id (key)
Output	station	Station_Id of the station where the function is executed or AM_UNKNOWN if the function has no assigned station.
Return		AM_OK, AM_FAILURE

2.3.10.8 Group directory interface

2.3.10.8.1 Type AM_GROUP

Definition	Type of an entry in the group directory	
Syntax	<pre> typedef UNSIGNED8 AM_GROUP; </pre>	
Usage	Groups are identified by a 6-bit address, represented by one octet, the two most significant bits being ignored.	

2.3.10.8.2 Procedure 'am_clear_groups'

Definition	Clears all entries in the group directory.	
Syntax	<pre> AM_RESULT am_clear_groups (void); </pre>	
Return		AM_OK, AM_FAILURE

2.3.10.8.3 Procedure 'am_insert_member'

Definition	Registers this Station_Id as a member into the group directory.	
Syntax	<pre> AM_RESULT am_insert_member (AM_GROUP group); </pre>	
Input	group	Group to which this station is to pertain.
Return		AM_OK, AM_FAILURE

2.3.10.8.4 Procedure 'am_remove_member'

Definition	Removes this Station_Id as a Group member.	
Syntax	<pre> AM_RESULT am_remove_member (AM_GROUP group); </pre>	
Input	group	Group from which this station is to be removed.
Return		AM_OK, AM_FAILURE

2.3.1.2 Procedure 'am_member'

Definition	Checks if the specified Station_Id is member of a Group.	
Syntax	<pre> AM_RESULT am_member (AM_GROUP group); </pre>	
Input	group	Group to which this Station_Id pertains.
Return		AM_OK if the Station_Id is member of the Group, AM_FAILURE otherwise.

2.3.10.9 Caller application interface

2.3.10.9.1 Procedure 'am_call_request'

Definition	Requests to send a Call_Message, sets up the data structures for receiving the Reply_Message and subscribes the indication procedures.	
Syntax	<pre> void am_call_request (UNSIGNED8 caller_function, const AM_ADDRESS * replier, void * out_msg_adr, UNSIGNED32 out_msg_size, void * in_msg_adr, UNSIGNED32 in_msg_size, UNSIGNED16 reply_timeout, AM_CALL_CONFIRM call_confirm, void * caller_ref); </pre>	
Input	caller_function	Function_Id of the Caller. AM_MANAGER_FCT shall be used if the Replier address is a System_Address.
	replier	Application address of the Replier function or station.
	out_msg_adr	pointer to the Call_Message to transmit.
	out_msg_size	total length in octets of the Call_Message.
	in_msg_adr	pointer to the buffer to put the Reply_Message.
	in_msg_size	maximal total length in octets of the accepted Reply_Message.
	reply_timeout	time-out value in multiples of 64 ms for the reply after the transfer of the Call_Message.
	call_confirm	pointer to the call confirmation procedure. 'call_confirm' will be called unless 'am_call_request' is cancelled successfully by am_call_cancel.
	caller_ref	External_Reference for the call to be returned by 'call_confirm'. It can be used in any way by the Caller.
Return		This procedure returns no value, since the result will be provided by 'call_confirm'.
Usage	1 - Before calling 'am_call_request', the procedures am_init and am_announce_device shall be called. 2 - If in_msg_adr is NULL, the Messenger allocates a buffer for the Reply_Message. The Caller is then responsible to return this buffer after use by calling am_buffer_free. 3 - The call is rejected if a conversation with the same Caller and Replier address (and in the same direction) is already established. 4 - A function directory entry for the Replier function shall be defined if the Station_Id of the Replier is AM_UNKNOWN. 5 - No bus communication takes place if Caller and Replier are within the same station. 6 - A call with a System_Address is rejected if the Function_Id is different from 254 (Manager). 7 - The Caller may not modify the message buffers until 'call_confirm' is called. 8 - The default reply time-out specified with am_announce_device is awaited if reply time-out is 0.	

2.3.10.9.2 Type 'AM_CALL_CONFIRM'

Definition	When a requested call completes, returning either an error status or the received Reply_Message, the Session layer shall call the procedure of the Caller, 'call_confirm', which is of this type.	
Syntax	<pre>typedef void (* AM_CALL_CONFIRM) (UNSIGNED8 caller_function, void * am_caller_ref, const AM_ADDRESS * replier, void * in_msg_adr, UNSIGNED32 in_msg_size, AM_RESULT status);</pre>	
Input	caller_function	Function_Id of the Caller
	replier	Application address of the Replier function or station.
	am_caller_ref	returned value which was specified in the related 'am_call_request'
	in_msg_adr	pointer to a buffer which contains the received Reply_Message. It is NULL if the Caller did not supply a buffer for the Reply_Message and if at the same time in_msg_size is 0.
	in_msg_size	total length in octets of the Reply_Message. It is 0 if an error occurred or if the Replier application only replies with a status.
	status	gives an error code < AM_MAX_ERR or on success the status supplied by the Replier.
Usage	1 - The procedure 'call_confirm' shall be previously subscribed by 'am_call_request'. 2 - AM_MANAGER_FCT is returned if the Replier address is a System_Address. 3 - Call confirmation implicitly returns the Call_Message buffer to the Caller. 4 - A Reply_Message buffer which was allocated by the Messenger shall be returned with am_buffer_free after use.	

2.3.10.9.3 Procedure 'am_call_cancel'

Definition	Cancels a call request which is not yet confirmed.	
Syntax	<pre>AM_RESULT am_call_cancel (UNSIGNED8 caller_function, const AM_ADDRESS * replier);</pre>	
Input	caller_function	Function_Id of the Caller
	replier	Application address of the called function or station.
Return		AM_OK cancel successful, AM_FAILURE any other error
Usage	The call confirmation procedure will not be called if the return value is AM_OK.	

2.3.10.10 Replier application interface

2.3.10.10.1 Procedure 'am_bind_replier'

Definition	Makes a Replier known to the Messenger and connects its procedures.	
Syntax	<pre> AM_RESULT am_bind_replier (UNSIGNED8 replier_function, AM_RECEIVE_CONFIRM receive_confirm, AM_REPLY_CONFIRM reply_confirm); </pre>	
Input	replier_function	Function_Id of the Replier to be bound.
	receive_confirm	receive confirmation procedure which will be called on completion of a receive request.
	reply_confirm	reply confirmation procedure which will be called on completion of a reply.
Return	AM_OK = 0 binding is successful, otherwise am_bind_replier performs no action. AM_ALREADY_USED this Replier function is already bound, the confirmation procedures are not modified. AM_NO_LOC_MEM_ERR no memory for the bind table, no Replier function can be bound. AM_FAILURE bind table is full. The size of the bind table is defined by am_announce_device.	
Usage	1 - 'am_init' and 'am_announce_device' shall be called before 'am_bind_replier'. 2 - Each Replier shall be bound before it can issue any 'am_receive_request'.	

2.3.10.10.2 Procedure 'am_unbind_replier'

Definition	Cancels all instances of the specified Replier and removes the binding.	
Syntax	<pre> AM_RESULT am_unbind_replier (UNSIGNED8 replier_function); </pre>	
Input	replier_function	Function_Id of the Replier to be unbound.
Return	AM_OK, AM_FAILURE	
Usage	Calls which have been received before calling am_unbind_replier, but have not yet been replied to, are cancelled.	

2.3.10.10.3 Procedure 'am_receive_request'

Definition	Informs that a Replier is ready to receive an incoming call.	
Syntax	<pre> AM_RESULT am_receive_request (UNSIGNED8 replier_function, void * in_msg_adr, UNSIGNED32 in_msg_size, void * replier_ref); </pre>	
Input	replier_function	Function_Id of the Replier expecting a call. Function_Id = 253 implies a System_Address.
	in_msg_adr	Pointer to a buffer for the incoming Call_Message. The Messenger allocates a buffer for the Call_Message if in_msg_adr is NULL. This buffer may not be modified until 'am_receive_request' is confirmed or cancelled.
	in_msg_size	Maximum total size in octets of the Call_Message which can be accepted.
	replier_ref	External reference returned with the related receive_confirm procedure. It is at the same time an instance reference and distinguishes the Replier instances which serve the same Replier.
Return	AM_OK The bound receive confirmation procedure will be called to pass a received call if the request is not cancelled. AM_ALREADY_USED the same Replier instance already issued a receive request which is not yet confirmed or cancelled or replied to. AM_FAILURE the Replier is not bound; AM_NO_LOC_MEM_ERR there is not enough memory to accept an 'am_receive_request'; AM_NR_OF_INST_OVF more simultaneous 'am_receive_request' have been issued than was limited by parameter 'max_inst_number' of 'am_announce_device'.	
Usage	This procedure requires that the procedure 'am_bind_replier' for the same Replier has been called previously.	

2.3.10.10.4 Type 'AM_RECEIVE_CONFIRM'

Definition	When the Session layer receives a Call_Message, it shall call the receive confirmation procedure receive_confirm, which is of this type.	
Syntax	<pre>typedef void (* AM_RECEIVE_CONFIRM) (UNSIGNED8 replier_function, const AM_ADDRESS * caller, void * in_msg_adr , UNSIGNED32 in_msg_size, void * replier_ref);</pre>	
Input	replier_function	Function_Id of the Replier as specified in the corresponding 'am_receive_request'.
	caller	Caller address
	in_msg_adr	Pointer to a buffer which contains the Call_Message.
	in_msg_size	Total length in octets of the received Call_Message.
	replier_ref	External reference as specified in the corresponding 'am_receive_request'.
Usage	1 -The receive confirmation procedure 'receive_confirm' has been previously subscribed by 'am_bind_replier'. 2 -If the Replier instance did not supply a buffer in 'am_receive_request', the reply buffer is supplied by the Messenger and the Replier shall return it after use with 'am_buffer_free'.	

2.3.10.10.5 Procedure 'am_reply_request'

Definition	Requests to send a Reply_Message in response to a previously received Call_Message	
Syntax	<pre> AM_RESULT am_reply_request (UNSIGNED8 replier_function, void * out_msg_adr, UNSIGNED32 out_msg_size, void * replier_ref AM_RESULT status); </pre>	
Input	replier_function	Function_Id of the Replier as specified in the corresponding 'am_receive_request'.
	replier_ref	External_Reference as specified in the corresponding 'am_receive_request'
	out_msg_adr	pointer to the Reply_Message buffer. This buffer shall not be modified until the reply request is confirmed. If out_msg_adr is NULL, only the status is transmitted to the Caller.
	out_msg_size	total length in octets of the Reply_Message.
	status	call execution result supplied by the Replier, transmitted to the Caller in addition to the Reply_Message itself.
Return		AM_OK, AM_FAILURE
Usage	1 -Each received call shall be replied with 'am_reply_request' or cancelled with 'am_receive_cancel'. 2 -This procedure returns before the Reply_Message is transmitted. 3 - The Messenger transmits this Reply_Message together with the specified status back to the Caller. 4 - The Caller address is retrieved internally from the Replier instance.	

2.3.10.10.6 Type 'AM_REPLY_CONFIRM'

Definition	When a Reply_Message has been completely sent and acknowledged by the Caller, or when any error occurred, the Session layer calls the reply confirmation procedure 'reply_confirm' which is of this type.	
Syntax	<pre>typedef void (* AM_REPLY_CONFIRM) (UNSIGNED8 replier_function, void * replier_ref);</pre>	
Input	replier_function	Function_Id of the Replier as specified in the corresponding 'am_receive_request'.
	replier_ref	External_Reference as specified in the corresponding 'am_receive_request'
Usage	1 - 'reply_confirm' has been previously subscribed by 'am_bind_replier'. 2 - This procedure returns the Reply_Message buffer back to the Replier instance. 3 - Reply confirmation allows a further 'am_receive_request' for the same Replier instance.	

2.3.10.10.7 Procedure 'am_receive_cancel'

Definition	Cancels an 'am_receive_request' or an 'am_reply_request' which is not yet confirmed, or announces that a received call will not be replied.	
Syntax	<pre>AM_RESULT am_receive_cancel (UNSIGNED8 replier_function, void * replier_ref);</pre>	
Input	replier_function	Function_Id of the Replier as specified in the corresponding 'am_receive_request'.
	replier_ref	External_Reference as specified in the corresponding 'am_receive_request'
Return	AM_OK, AM_FAILURE	
Usage	1 - The confirmation procedure for a successfully cancelled 'reply request' will not be called any more. 2 - It is up to the user to distinguish if the receive request has completed (i.e. a Call_Message has still been received) or not and to deallocated dynamic buffers for the Call_Messages.	

2.3.10.11 Procedure 'am_buffer_free'

Definition	Deallocates a message buffer previously allocated by the Session layer after use.	
Syntax	<pre> AM_RESULT am_buffer_free (void * in_msg_adr, UNSIGNED32 size); </pre>	
Input	in_msg_adr	pointer to the released buffer.
	size	total length in octets of this buffer.
Return	AM_OK, AM_FAILURE	
Usage	Buffer allocation is independent from the packet pool management.	

2.3.10.12 Multicast Application Interface

The application interface for multicast messages shall be identical to that for single-cast messages.

The replier is not expected to send back a Reply_Message, but it is expected to call 'am_reply_request' to deallocated a dynamic buffer if such is used. No Reply_Message, however, shall be generated in this case.

2.4 Presentation and encoding of transmitted and stored data**2.4.1 Purpose**

This subclause specifies data types and defines an abstract syntax notation to express these data types and the encoding rules used for transmission or storage. This notation is based on ASN.1 (ISO/IEC 8824), but includes additional constructs suited for real-time communication. By contrast, the internal interfaces in a device are specified in the 'C' language, which is less precise.

2.4.2 Data ordering**2.4.2.1 Transmission format**

This Standard prescribes the order in which bits and words are transmitted over the Train Communication Network. To this effect, it defines a number of primitive types and structured types. The meaning of the data is outside the scope of this standard.

2.4.2.2 Traffic_Store format

This standard recommends to store data in the Traffic_Store in the same format as they are transmitted over the bus, treating them as an array of octets.

2.4.2.3 Application data format

This Standard does not specify the Application data format. The interface procedures are expected to convert the data formats of the Application to theses used for storage or transmission and vice-versa.

2.4.2.4 General rules

- a) Data structures shall be numbered from left to right and from top to bottom, in the order of reading an english text. The first item on the top and to the left has offset zero.
- b) Memory shall be treated as an array of octets, and shall be transmitted in order of increasing address, regardless of the size of the transmitted units (by octets, by 32-bit words, etc.). The first octet has octet offset zero.
- c) Bits within a data structure shall be identified by their offset with respect to the beginning of the structure. If this structure would contain an unsigned integer, the most significant bit of that integer would have offset zero. This bit is considered to be the 'leftmost' one when reading the data structure.
- d) All data shall be transmitted most significant octet first (Big-Endian)
- e) The order of bit transmission within an octet is considered a bus issue, invisible to the programmer. For instance, HDLC protocols such as the WTB uses first transmit the least significant bit of an octet (offset 7), while the MVB transmits it last.
- f) Information about the data type is not sent with the data. Types are expected to be defined and agreed beforehand between the users of the TCN in a specific application.
- g) The elements of a structured type (Record, Sequence) shall be transmitted in the order they are declared.
- h) Arrays shall be transmitted in order of increasing index. Multi-dimensional arrays are transmitted in the order their indices are listed (e.g. ARRAY OF [row, column] is transmitted row by row).
- i) To ease implementation, a variable shall be stored at an offset address which is a multiple of its size (alignment).
- j) Variable length data (open arrays, records, sets,...) shall not be used as Process_Variables, but may be transmitted in messages.

2.4.2.5 Relationship with ASN.1

The ISO/IEC 8824 defines the Abstract Syntax Notation One (ASN.1) to express data structures in machine-readable form.

Although ASN.1 does not impose a transfer syntax, it cannot express the compact encoding rules frequently used by programmers where bandwidth or where time is limited. In addition, it cannot express already existing encodings which do not obey to its structuring method.

Therefore, this standard defines an abstract syntax based on ASN.1, which expresses at the same time the data encoding and the bit-by-bit content of the data.

The following keywords have been added to ASN.1:

ALIGN	ANTIVALENT2	ARRAY	BCD4
BIPOLAR2.16	BIPOLAR4.16	BITSET#	BITSET_L#
BOOLEAN1	BOOLEAN8	ENUM#	INDIRECT
INTEGER#	INTEGER_L#	ONE_OF	REAL32
REAL64	RECORD	SOME_OF	STOP
STRING#	TIME64	TIMEDATE48	UNICODE16
UNIPOLAR2.16	UNSIGNED#	UNSIGNED_L#	WORD#

This notation uses compact encoding rules:

- it assumes that all user-defined types are recognised by the destination.
- it uses primitive types of fixed size (in ASN.1, an integer can be of any size)
- it includes the size of the elements explicitly in a dedicated field where needed.
- it introduces own keywords to avoid confusion with ASN.1 where the semantic is similar, but different (for instance ONE_OF instead of CHOICE, SOME_OF instead of SET).
- it uses no implicit type-tagging, except for the ONE_OF and SOME_OF types, where tagging is explicitly done by a dedicated field.
- it has no optional fields (except in SOME_OF).
- it is not aligned, although alignment can be specified.

The following rules for notation are used:

- Keywords, including basic types, and constant identifiers are entirely in upper case.
- Type identifiers begin with an Uppercase letter
- Field identifiers begin with a lower case letter

2.4.3 Notation for the primitive types

2.4.3.1 Notation for the boolean type

2.4.3.1.1 Definition

A primitive type with two distinguished values, TRUE and FALSE.

NOTE 1 This is the ASN.1 definition of a 'BooleanType'.

NOTE 2 This type is used to represent binary inputs and outputs (relay, led, micro switch, etc.).

2.4.3.1.2 Syntax

BooleanType ::= BOOLEAN1

2.4.3.1.3 Encoding

A variable of boolean type shall be encoded as one bit:

1st	interpretation
0	FALSE
1	TRUE

2.4.3.2 Notation for the antivalent type

2.4.3.2.1 Definition

A primitive type with four distinguished values.

NOTE 1 This is not an ASN.1 type.

NOTE 2 Variables of this type are used as check variable for other variables or for critical Booleans.

2.4.3.2.2 Syntax

AntivalentType ::= ANTIVALENT2

2.4.3.2.3 Encoding

A variable of antivalent type shall be transmitted as 2 bits, the first corresponding to the boolean meaning of the variable and the second to its inverse.

It may take one of four states, as follows:

0	1	interpretation
2^{+1}	2^0	
0	0	ERROR
0	1	FALSE
1	0	TRUE
1	1	UNDEFINED

NOTE the ERROR and UNDEFINED states may be interpreted as legal states by an application.

2.4.3.3 Notation for the unsigned integer types

2.4.3.3.1 Definition

A primitive type with distinguished values which are positive whole numbers, including zero (as a single value), having a fixed size in bits defined by the postfix #.

NOTE This is a ASN.1 'IntegerType', restricted to a fixed size # and non-negative values.

2.4.3.3.2 Syntax

UnsignedType ::= UNSIGNED#, (# = any unsigned integer).

2.4.3.3.3 Encoding

An unsigned integer shall be transmitted in binary representation, most significant bit first.

When the carried value has a smaller size than the UNSIGNED# type, it shall be right-justified and extended to the left with zeroes.

2.4.3.3.3.1 UNSIGNED8 encoding

0	1	2	3	4	5	6	7
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Range: 0..255

2.4.3.3.3.2 UNSIGNED16 encoding

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

Range: 0..65535

2.4.3.3.3.3 UNSIGNED32 encoding

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

Range: 0..+2³²-1

2.4.3.4 Notation for the integer type

2.4.3.4.1 Definition

A primitive type with distinguished values which are positive and negative whole numbers, including zero (as a single value), having a fixed size in bits defined by the postfix #.

NOTE This is a ASN.1 'integer type', restricted to a fixed size of #.

2.4.3.4.2 Syntax

IntegerType ::= INTEGER#, (# = any unsigned integer).

2.4.3.4.3 Encoding

The value shall be represented in binary 2's complement, with the first transmitted bit being the sign bit.

When the carried value has a smaller size than the INTEGER# type, it shall be right-justified and sign-extended to the left (if it is negative, with '1', otherwise, with '0')

2.4.3.4.3.1 INTEGER8 encoding

0	1	2	3	4	5	6	7
sign	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Range: -128 .. +127

EXAMPLE: '1111 1110'B = - 2

2.4.3.4.3.2 INTEGER16 encoding

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

Range: $-2^{15} .. 2^{15} - 1$

2.4.3.4.3.3 INTEGER32 encoding

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

Range: $-2^{31} .. 2^{31} - 1$

2.4.3.5 Notation for the enumerated type

2.4.3.5.1 Definition

A primitive type whose values are given distinct identifiers as part of the type notation, having a fixed size in bits defined by the postfix #.

NOTE This is an ASN.1 ENUMERATED type, restricted to a fixed size of #.

2.4.3.5.2 Syntax

EnumeratedType ::= ENUM#{Enumeration}

with

(# = any unsigned integer)

Enumeration ::= NamedNumber | Enumeration, NamedNumber

and

NamedNumber ::= identifier (UnsignedNumber) | identifier (DefinedValue)

Values can be listed in any order.

EXAMPLE:

```
Day_Of_Week_Type ::= ENUM4
{
    monday      (1),
    tuesday     (2),
    wednesday   (3),
    thursday    (4),
    friday      (5),
    saturday    (6),
    sunday      (7),
    undefined    (0)
}
```

Value '2' means 'TUESDAY'.

2.4.3.5.3 Encoding

Values of ENUM# shall be represented by an unsigned integer occupying the same place.

2.4.3.5.3.1 ENUM4 encoding

0	1	2	3
2^3	2^2	2^1	2^0

Range: 0..15

EXAMPLE: '0001'B means 'Monday' in the above example.

2.4.3.5.3.2 ENUM8 encoding

0	1	2	3	4	5	6	7
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Range: 0..255

EXAMPLE: '0000 0001'B means 'Monday' in the above example (considering it is ENUM8 rather than ENUM4).

2.4.3.6 Notation for the binary coded decimal type

2.4.3.6.1 Definition

A 4-bit unsigned integer expressing a decimal digit between 0 and 9.

NOTE This type does not exist in ASN.1

2.4.3.6.2 Syntax

BinaryCodedDecimalType ::= BCD4

2.4.3.6.3 Encoding

A BCD4 shall be encoded as an unsigned integer occupying the same place.

0	1	2	3
2^3	2^2	2^1	2^0

Range: 0..9 (other values undefined)

EXAMPLE: '0111'B = 7.

NOTE some undefined values may be used, for instance to designate the sign or another arithmetic operator.

2.4.3.7 Notation for the unipolar types

2.4.3.7.1 Definition

primitive types with distinguished values which are non-negative, whole numbers divided by a fixed power of two, expressing a value in percent of a span.

NOTE These types do not exist in ASN.1, they are expressed in IEC 870 as 'unsigned fixed point number'

2.4.3.7.2 Syntax

UnipolarType ::= UNIPOLAR2.16

NOTE 1 The number before the comma gives the number of power of 2 forming the integer part

NOTE 2 The epsilon factor is equal to the value of the smallest power of two in the word.

2.4.3.7.3 Encoding

A variable of unipolar type shall be transmitted as an unsigned integer.

2.4.3.7.3.1 UNIPOLAR2.16 encoding

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}
integer part		fractional part													

Span: 0 .. 400% - epsilon

2.4.3.8 Notation for the bipolar types

2.4.3.8.1 Definition

Primitive types with distinguished values which are positive or negative, whole numbers (including zero) divided by a fixed power of two, expressing a value in percent of a span.

NOTE These types do not exist in ASN.1, they are expressed in IEC 60870-5-1 as 'signed fixed point number'

2.4.3.8.2 Syntax

BipolarType ::= BIPOLAR2.16 | BIPOLAR4.16

NOTE 1 The number before the comma gives the number of power of 2 forming the integer part.

NOTE 2 The epsilon factor is equal to the value of the smallest power of two in the word.

2.4.3.8.3 Encoding

2.4.3.8.3.1 BIPOLAR2.16 encoding

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sign	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}
integer part		fractional part													

Span: -200%..+200%-epsilon

2.4.3.8.3.2 BIPOLAR4.16 encoding

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sign	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}
integer part				fractional part											

Span: -800%..+800%-epsilon

2.4.3.9 Notation for the real type

2.4.3.9.1 Definition

A primitive type whose distinguished values are members of the set of real numbers.

2.4.3.9.2 Syntax

RealType ::= REAL32

2.4.3.9.3 Encoding

This type shall be encoded as IEEE 754 prescribes for Short Real Number (32-bit).

NOTE 1 This is an ASN.1 'RealType', restricted to the IEEE 754 Short Real Number format.

NOTE 2 The 64-bit floating point number of IEEE 754 (REAL64) is not considered useful in this context.

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

Range: $\pm 3.37 \times 10^{+38}$

2.4.3.10 Notation for the character type

2.4.3.10.1 Definition

A primitive type whose distinguished values are members of the set of characters defined in ISO/IEC 8859-1.

2.4.3.10.2 Syntax

CharacterType ::= CHARACTER8

2.4.3.10.3 Encoding

Characters shall be transmitted in one octet, without parity bit.

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

EXAMPLE - '01100001'B = character 'a' according to ISO/IEC 8859-1.

2.4.3.11 Notation for the Unicode character type

2.4.3.11.1 Definition

A primitive type whose distinguished values are members of the set of characters defined in ISO/IEC 10646-1.

2.4.3.11.2 Syntax

UnicodeType ::= UNICODE16

2.4.3.11.3 Encoding

Unicode characters shall be transmitted in two octets.

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

2.4.3.12 Notation for the uncommitted types

2.4.3.12.1 Definition

An uncommitted type of undefined contents, but of fixed size.

2.4.3.12.2 Syntax

AnyType ::= WORD#, (# = any unsigned integer)

2.4.3.12.3 Encoding

A variable of uncommitted type has no prescribed encoding.

Bits shall be named according to the power of two of a variable of type UNSIGNED# which would occupy that place.

NOTE This naming is in the reverse direction as the offset within the same word.

2.4.3.12.3.1 WORD8 encoding

0	1	2	3	4	5	6	7
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

2.4.3.12.3.2 WORD16 encoding

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

2.4.4 Structured types

2.4.4.1 General

Five different structured types are defined:

- RECORD (variable length),
- ARRAY (fixed length or variable length),
- BITSET# (fixed length),
- ONE_OF (variable length)
- SOME_OF (variable length).

2.4.4.2 Notation for the record types

2.4.4.2.1 Definition

A structured type defined by referencing a fixed, ordered list of types; each value of the new type is an ordered list of values, one from each of the component types

NOTE 1 This type is an ASN1 'Sequence Type' with no optional types.

NOTE 2 It is recommended to observe alignment when defining a RECORD, i.e. all elements should be located at an offset with respect to the beginning of the record which is a multiple of their size.

2.4.4.2.2 Syntax

RecordType ::= RECORD { ElementTypeList }

with

ElementTypeList ::= ElementType | ElementTypeList, ElementType
and

ElementType ::= identifier Type | Type

The elements of a RECORD shall be identified by the identifier of the RECORD field followed by a dot and the subfield identifier, which may itself be a structured type.

EXAMPLE:

file.date.day

2.4.4.2.3 Encoding

Elements of a RECORD shall be transmitted in the order of their declaration.

EXAMPLE - A value of type Date32 is represented as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
year															
dummy				month				day							

```
Date32 ::= RECORD
{
  year      INTEGER16,
  dummy     WORD4,
  month     UNSIGNED4,
  day       UNSIGNED8
}
```

'dummy' was introduced to align the variable 'day' on a 16-bit word boundary.

2.4.4.3 Notation for the bitset types

2.4.4.3.1 Definition

An ARRAY [#] of BOOLEAN1, having a fixed size in bits defined by the postfix #.

NOTE This type corresponds to BITSTRING in ASN.1.

2.4.4.3.2 Syntax

BitsetType ::= BITSET# {NamedBitList}

with

NamedBitList ::= NamedBit | NamedBitList, NamedBit

and

NamedBit ::= identifier (number) | identifier (DefinedValue)

- a) The value of each 'number' or 'DefinedValue' appearing in the 'NamedBitList' shall be different, and is the offset of a distinguished bit in a bitset value.
- b) Each 'identifier' appearing in the 'NamedBitList' shall be different.
- c) All elements are implicitly of type BOOLEAN1. The DefinedValue can only be TRUE (1) or FALSE (0).
- d) Elements shall be declared in order of increasing offset.
- e) If all elements of the BITSET are declared, 'number' can be omitted. This should be the normal case.

2.4.4.3.3 Encoding

Elements of a bitset shall be transmitted in order of declaration.

NOTE The name of the bitset elements may be different from their offset. In assembly language programming, the bit with offset 0 is usually named 'B7' in an 8-bit computer, or 'B15' in a 16-bit computer.

2.4.4.3.3.1 BITSET8 encoding

0	1	2	3	4	5	6	7
1st							8th

Range:

8-bit Set of Boolean

EXAMPLE:

```
AccessType8 ::= BITSET8
{
  system      (0),    -- first bit of the bitset (MSB)
  owner       (1),
  group       (2),
  world       (3),
}
```

is equivalent to:

```
AccessType8 ::= BITSET8
{
  system,          -- first bit of the bitset (MSB)
  owner,
  group,
  world,
  reserved4
  reserved5
  reserved6
  reserved7        -- 8th or last bit of the bitset (LSB)
}
```

An UNSIGNED8 occupying that space with a value of '80'H means that 'system' is the only member of the set.

2.4.4.3.3.2 BITSET16 encoding

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1st															

EXAMPLE:

AccessType ::= BITSET16 { system (0), owner (1), group (2), world (3)}

Value '0110 0000 0000 0000'B means that 'owner' and 'group' are members of the set.

2.4.4.3.3.3 BITSET32 encoding

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

2.4.4.3.3.4 BITSET64 encoding

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1st															

2.4.4.4 Notation for the array type**2.4.4.4.1 Definition**

A structured type, defined by referencing a single existing type; each value of the new type is an ordered list of zero, one or more values of the existing type. The position of each value is identified by an index. The number of values is indicated by either a constant or a field of the embedding structure. The number of values may be omitted if a stop element is supplied.

NOTE An ARRAY is an ASN1 'SequenceOf Type' with a number of elements indicated by a constant, a dedicated variable or not at all (stop element).

2.4.4.4.2 Syntax

ArrayType ::= ARRAY [IndexList] OF Type

IndexList ::= Index | IndexList, Index

Index ::=

number | DefinedValue |
 identifier |
 identifier UnsignedType |
 UnsignedType |
 STOP = Value

The number, DefinedValue or identifier specify the size of the array in number of elements (0 for a void array). Its type shall be an unsigned integer.

If the identifier names a field declared outside of the array, this field shall be located in the embedding data structure at the same level of nesting, or be a subfield of a field located at the same level of nesting, in which case the full path name shall be indicated.

The size may be given by an arithmetic expression.

2.4.4.4.3 Encoding

Multi-dimensional arrays shall be transmitted in the order their indices are listed.

Arrays of octets (uncommitted contents, e.g. memory dump) shall be transmitted in increasing memory address (or index) of the Application Memory.

EXAMPLES

```
DumpOctetType ::= ARRAY [octet_count UNSIGNED16] OF WORD8.
```

2 - Transmission of the same memory dump by words of 16 bits, 'word_count' having half the value of 'octet_count' of the preceding example:

```
DumpWordType ::= ARRAY [word_count UNSIGNED16] OF WORD16.
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
word_count															
word at address M															
word at address (M + word_count x 2 -2)															

3 - Number of elements given by a field in the nesting data structure (at an unspecified offset):

```
DumpOctetType ::= ARRAY [array_count] OF WORD8.
```

4 - Number of elements given by a field of a structured value in the nesting structure:

HeaderType ::= RECORD

```
{
name          ARRAY [32] OF CHARACTER8
bodysize      UNSIGNED16,
...
}
```

FrameType ::= RECORD

```

{
header      HeaderType
body       ARRAY [ header.bodysize ] OF CHARACTER8
}

```

5 - Character strings, in which the stop character is a 'space':

```
ProfibusString ::= ARRAY [STOP = '20'H] OF CHARACTER8.
```

2.4.4.5 Notation for the choice types

2.4.4.5.1 Definition

A structured type, defined by referencing a fixed, unordered, list of distinct types; each value of the new type is a value of (exactly) one of the component types.

NOTE This type corresponds to the ASN1 'ChoiceType' but has a dedicated tag.

2.4.4.5.2 Syntax

```
OneOfType ::= ONE_OF [identifier identifier EnumeratedType]
                                     {AlternativeTypeList}
```

with

~~AlternativeTypeList ::= ElementType | ElementTypeList, ElementType~~

and

```
ElementType ::= identifier [tag] Type | [tag] Type
```

and

```
tag ::= UnsignedNumber | DefinedValue | identifier
```

If a named variable is used as a tag, this variable shall be located in the structure embedding the element.

If the tag variable is located at the same nesting level as the choice, only the name of the variable shall be included.

If the variable is at another level of nesting, the path to the same level of nesting shall be included.

EXAMPLES:

1 - The tag is a number (not recommended since this number shall be defined in different places):

```
Commands ::= ONE_OF [choice_var ENUM8]
{
  [3]                OpenSequence,
  [2]                CloseSequence,
  [5]                StandbySequence
}
```

2 - The tag is an enumeration type located in the 16 bits before the choice:

```
CommandType ::= ENUM16
```

```
{
  OPEN                (3),
  CLOSE                (2),
  STANDBY              (5)
}
```

```
Commands ::= ONE_OF [choice_var CommandType]
```

```
{
  [OPEN]              OpenSequence,
  [CLOSE]              CloseSequence,
  [STANDBY]            StandbySequence
}
```

3 - The tag is defined at the same level of nesting in the embedding structure:

```
Commands ::= ONE_OF [choice_var]
```

```
{
  [OPEN]              OpenSequence,
  [CLOSE]              CloseSequence,
  [STANDBY]            StandbySequence
}
```

```
Command_Frame ::= RECORD
```

```
{
  choice_var          CommandType,
  ...
  command              Commands;
  ...
}
```

4 - The tag is defined in a subfield of a field located at the same level of nesting:

```
Commands ::= ONE_OF [Command_Frame.header.choice_var]
```

```
{
  [OPEN]              OpenSequence,
  [CLOSE]              CloseSequence,
  [STANDBY]            StandbySequence
}
```

```
Command_Frame ::= RECORD
```

```
{
  header              RECORD
  {
    ....addresses      ...
    choice_var          CommandType
    ....
  }
  commands              Commands
}
```

NOTE Relative paths (e.g. -/header) are not recommended.

2.4.4.5.3 Encoding

A ONE_OF shall be encoded by transmitting before the value the tag field indicating which choice has been made.

The size of the transmitted value is either implicit or indicated in the type itself.

NOTE A ONE_OF is a SOME_OF with only one element.

EXAMPLE - A particular value of the above Commands choice will be transmitted as:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
choice_var (= 5)															
first octet of (StandbySequence)															
								last octet of (StandbySequence) or dummy							

2.4.4.6 Notation for the set types

2.4.4.6.1 Definition

A structured type, defined by referencing a fixed, unordered list of distinct types, some of which may be declared as optional; each value of the new type is an unordered list of values, one for each of the transmitted component types.

NOTE This type corresponds to the ASN1 'SetType', but has an explicit tag.

2.4.4.6.2 Syntax

SetType ::= SOME_OF { ElementTypeList }

with

ElementTypeList ::= ElementType | ElementTypeList, ElementType

and

ElementType ::= [tag] NamedType

and

tag ::= identifier | identifier ElementType | ElementType

If the tag is a named variable, the corresponding variable shall belong to a data structure at the same level of nesting or belong to a subfield of a field at the same level of nesting, in which case it shall be identified by its full path name.

If the members of the set type are fixed in number, the reference name can be omitted to every member whose purpose is evident from its type.

If the selector an enumerated type, the enumeration constants used to select the set elements shall be put in bracket.

If a bitset variable is used as a selector, this variable shall be defined previously within the embedding data structure or belong to a subfield of a field at the same level of nesting, in which case it shall be identified by its full path name.

EXAMPLES

1 – Tag as an unsigned integer in the field preceding the set value

```
MemberType ::= SOME_OF [UNSIGNED8]
{
    OPENSEQ          [3]          Type_OpenSequence,
    CLOSESEQ         [2]          Type_CloseSequence,
    STANDBY           [5]          Type_StandbySequence
}
```

2 - Omission of the reference name

```
MemberType ::= SOME_OF [UNSIGNED8]
{
    [3]              Type_OpenSequence,
    [2]              Type_CloseSequence,
    [5]              Type_StandbySequence
}
```

3 – Enumeration type as tag (recommended practice)

```
MemberType          ENUM8
{
    OPENSEQ          (3),
    CLOSESEQ         (2),
    STANDBY           (5)
}
...
CommandsType ::= SOME_OF [MemberType]
{
    [OPENSEQ]        Type_OpenSequence,
    [CLOSESEQ]        Type_CloseSequence,
    [STANDBY]         Type_StandbySequence
}
```

4 – Use of a bitset as tag

```
MembersType          BITSET8
{
    OPENSEQ          (3),
    CLOSESEQ         (2),
    STANDBY           (5)
}
...
CommandsType ::= SOME_OF [members]
{
    [OPENSEQ]        Type_OpenSequence,
    [CLOSESEQ]        Type_CloseSequence,
    [STANDBY]         Type_StandbySequence
}
```

```
Commands_Frame ::= RECORD
{
    ...
    members          MembersType,
    ...
    commands          CommandsType
    ...
}
```

2.4.4.6.3 Encoding

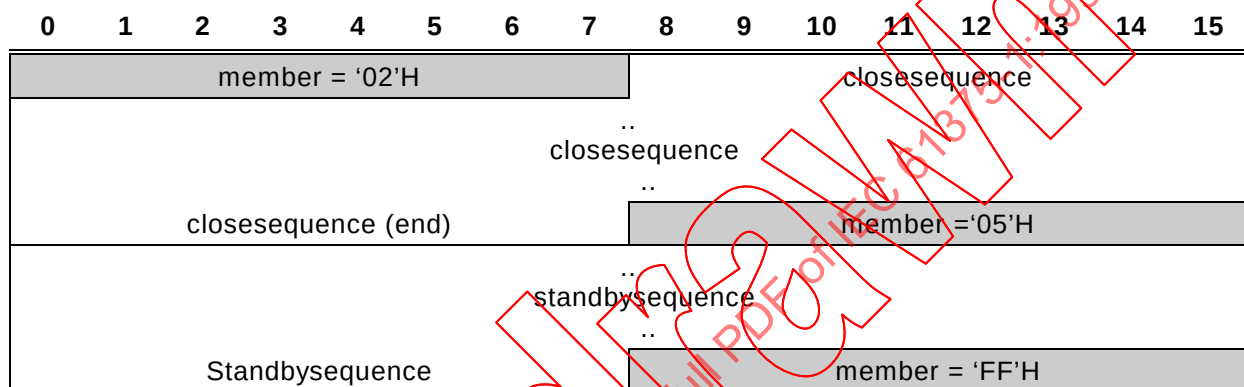
A set shall be encoded by transmitting each chosen value.

If a tag is contained, it shall precede each selected value, the particular tag value 'FF'H (all ones) closing the transmitted set.

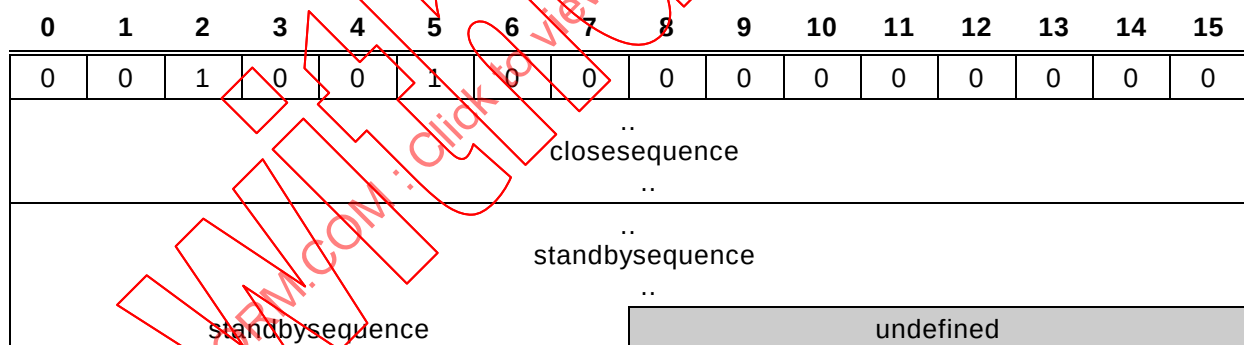
If the tag is replaced by a bitset, the bitset shall be transmitted before the set and the different members of the set shall be transmitted contiguously.

EXAMPLES

1 - A particular value of the above MemberType set will be transmitted as:



2 - If the tag is replaced by a bitset, the coding will be:



2.4.5 Alignment

In any type, it may be necessary to add padding bits to align the next field on a 16 or a 32-bit boundary (or any other boundary). To express this, the qualifier ALIGN is used after the type. The padding bits are not defined (they are 0 by default).

EXAMPLE - The following defines an array of characters which is aligned on a 32-bit boundary, regardless of the value of 'count'.

AlignedString ::= ARRAY ALIGN 32 [count] OF CHARACTER8.

2.4.6 Notation for special types

Some structured types have a special type designator.

2.4.6.1 Notation for the string type

STRING# is an ARRAY [] OF CHARACTER8, in which the stop element shall be the character '00'H, the actual size of the string is deduced from the number of significant characters, although the number of transmitted characters may be larger

EXAMPLE - A text string of type STRING32 is represented by an ARRAY [32 STOP='00'H] OF CHARACTER8.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1st character or '00'H								2nd character or '00'H							
								characters ...							
last character or '00'H								32nd character or '00'H							

2.4.6.2 Notation for the TIMEDATE48 type

2.4.6.2.1 Definition

A structured type expressing the absolute time in number of seconds since Universal Co-ordinated Time (UTC), 00:00:00, 1st January 1970 (Unix and ANSI-C format).

NOTE This type is used for distribution of the actual time, event tagging, synchronisation.

2.4.6.2.2 Syntax

```
TimeDate48 ::= RECORD
{
  seconds      UNSIGNED32,      -- elapsed since 1970, January
                                1st, 00:00
  ticks        UNSIGNED16      -- fraction of seconds (1 tick =
                                1/65536s)
}
```

2.4.6.2.3 Encoding

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
seconds (most significant)															
seconds (least significant)															
ticks = 1/65536 s															

Time can be represented to a granularity of 15,3 µs (=1/65536 s).

The range is 136 years.

The precision of the fractional part shall be at least 10 bits.

Unused low order bits shall be set to zero.

NOTE A TimeDate48 variable will wrap around on year 2038, January 19, 3:14:07 UTC. This wrap-around should be considered in the test of the software.

2.4.6.3.1 Definition

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
seconds (most significant)															
seconds (least significant)															
ticks = 1/65536 s															
chirp = 0,232 μ s															

[illegible]

2.4.6.5 Notation for Little-Endian types

For compatibility with foreign devices, the following types allow to express data structures in Little-Endian notation. These types should not be used in new projects:

ENUM_L16 Little-Endian representation of a 16-bit enumeration type

INTEGER_L16 Little-Endian representation of a 16-bit integer

INTEGER_L32 Little-Endian representation of a 32-bit integer

UNSIGNED_L16 Little-Endian representation of a 16-bit unsigned

UNSIGNED_L32 Little-Endian representation of a 32-bit unsigned

NOTE Other types may be constructed on the same pattern.

IECNORM.COM: Click to view the full PDF of IEC 61375-1:1999

3 Multifunction Vehicle Bus

This clause applies where MVB is required.

A tutorial on the MVB is given in Clause A.3.

3.1 General

3.1.1 Contents of this clause

This clause specifies one component of the Train Communication Network, the Multifunction Vehicle Bus (MVB), a serial data communication bus designed primarily, but not exclusively, for interconnecting equipment where interoperability and interchangeability are needed.

This clause specifies:

- a) the physical media in single-line and double-line configurations;
- b) the signalling and the redundancy handling;
- c) the format and timing of the transmitted frame and telegrams;
- d) the organisation of the bus traffic;
- e) the allocation of Mastership;
- f) the management of the bus;
- g) the Link Layer interface and the layer management interface.

3.1.2 Structure of this clause

This clause is structured following the OSI layers of a reference MVB device as shown in Figure 43:

Subclause 3.1 General

Subclause 3.2 Physical Layer

- Electrical medium for short distance (RS-485, 20,0 m)
- Electrical medium for middle distance (transformer-coupled, 200,0 m)
- Optical fibre for long distances (glass fibres, 2,0 km)

Subclause 3.3 Medium-dependent signalling

- Frame encoding and decoding
- Line Unit interface
- Physical redundancy handling

Subclause 3.4 Frames and Telegrams

- Master Frame and Slave Frame encoding, Telegram timing

Subclause 3.5 Link Layer Control

- Addressing
- Master Frame and Slave Frame format

Subclause 3.6 Medium allocation

- Periodic Polling
- Event Polling
- Devices Scan

Subclause 3.7 Mastership transfer

- Regular and exceptional mastership transfer

Subclause 3.8 Link Layer Interface

- Link Process Data Interface (LPI),
- Link Message Data Interface (LMI),
- Link Supervision Interface (LSI).

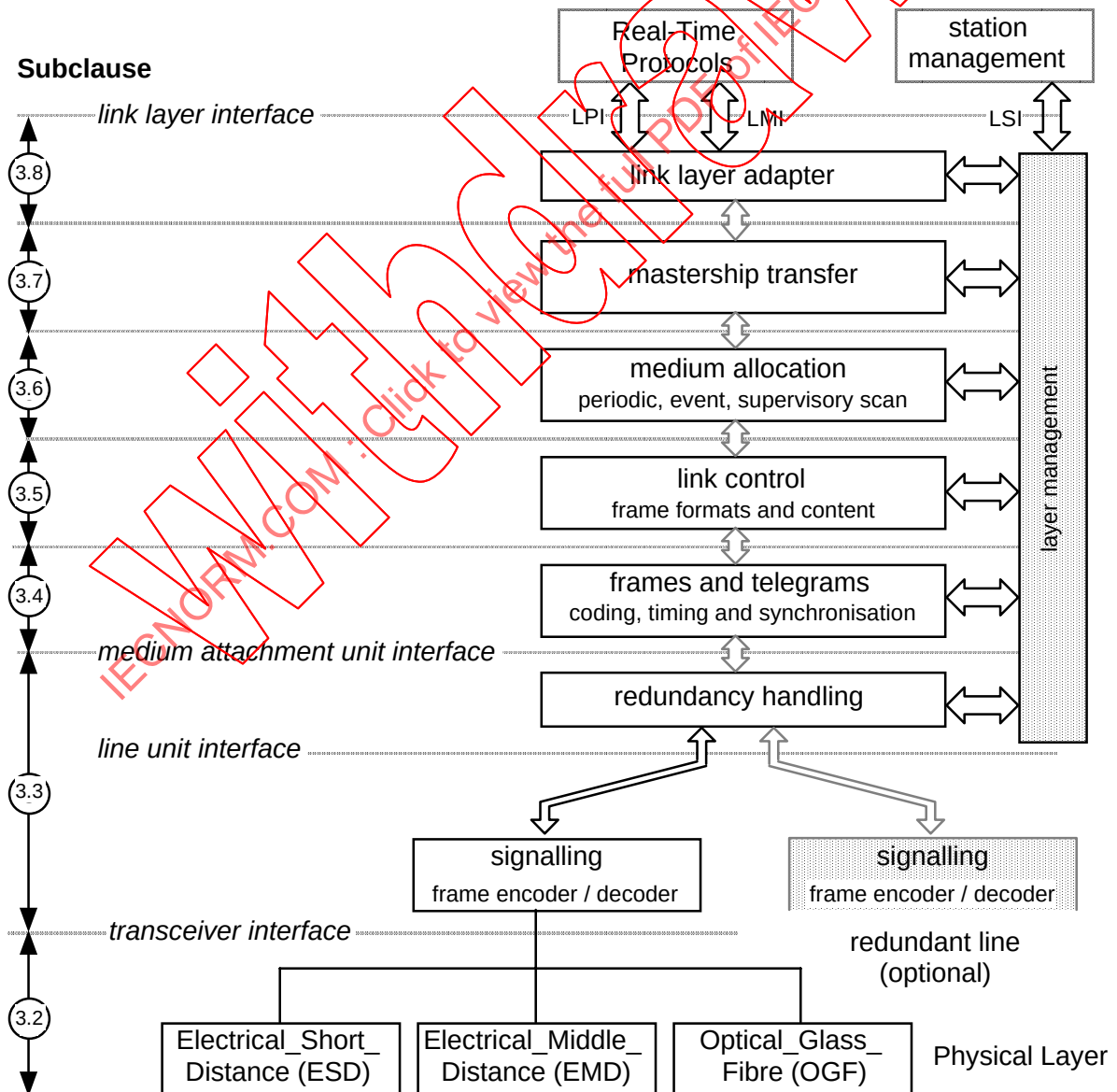


Figure 43 - Reference device and structure of the document

3.2 Physical Layer

3.2.1 Topology

3.2.1.1 Segments

An MVB shall consist of one or several bus segments of the following media:

- a) ESD: an Electrical Short Distance medium using a pair of wires with differential transmission according to the RS-485 standard, supporting up to 32 devices over a distance of 20,0 m without need for galvanic separation, and over longer distances when using galvanic isolation, as specified in 3.2.4;
- b) EMD: an Electrical Middle Distance medium, using a shielded, twisted wire pair, supporting up to 32 devices over a distance of 200,0 m and allowing transformers for galvanic isolation, as specified in 3.2.5;
- c) OGF: an Optical Glass Fibre medium, using optical fibres bussed through a star coupler, over a distance up to 2,0 km, intended for critical environments such as locomotives, as specified in 3.2.6.

3.2.1.2 Couplers

MVB segments shall be interconnected by couplers of one of the following types:

- a) repeater for connecting different media;
- b) star couplers for forming optical fibres into a bus.

EXAMPLE - An MVB configuration, shown in Figure 44, consists of an ESD segment supporting a bus administrator, several devices within and outside of a rack and a gateway, an EMD segment supporting several devices and a bus administrator, and an OGF segment supporting a number of devices. Segments are interconnected by repeaters. The star coupler consists of an ESD segment which, in this example, supports no devices. Each segment in Figure 44 could be duplicated to increase availability.

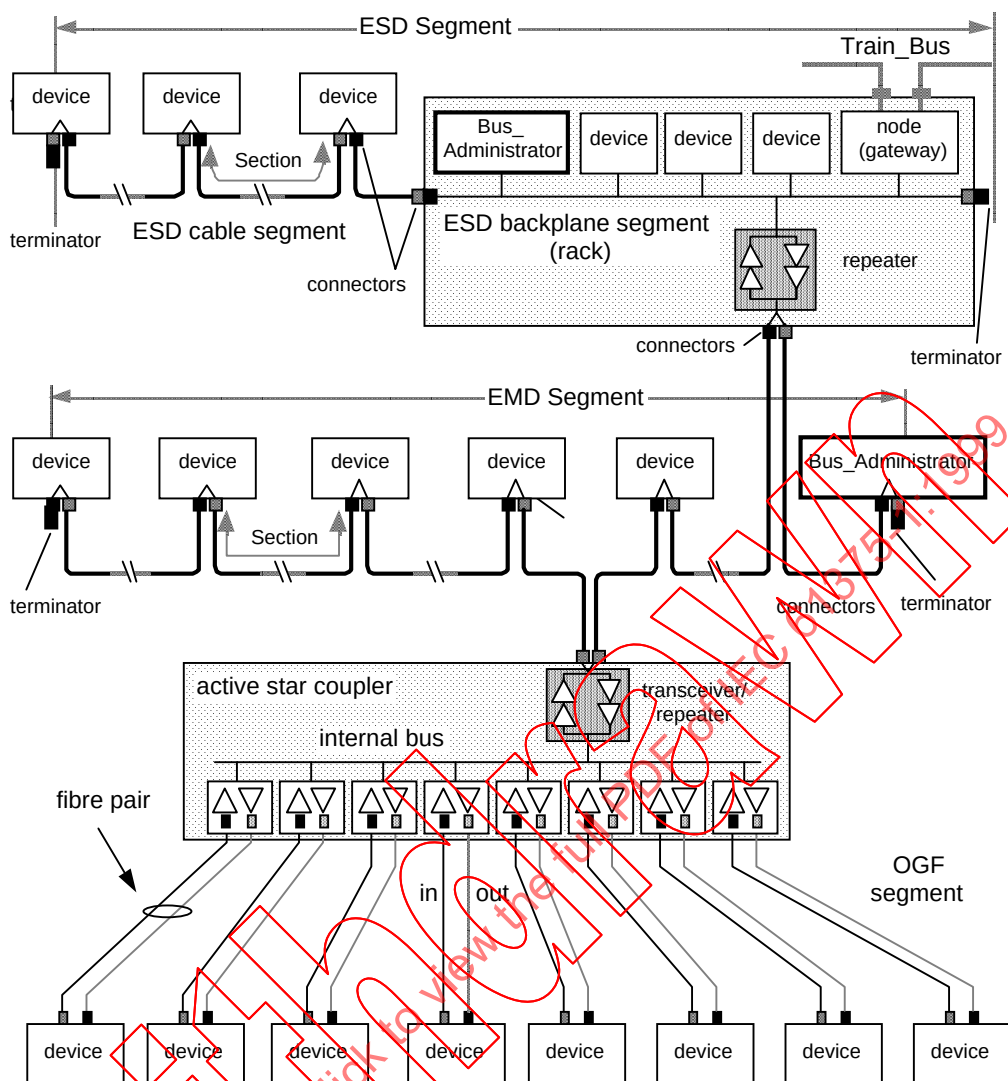


Figure 44 - MVB configuration

3.2.1.3 Double-line segments

As an option, a segment may be duplicated to increase availability (double-line segment).

An MVB may consist of a combination of double-line segments and single-line segments interconnected by bus couplers.

3.2.2 Device classes

3.2.2.1 Capabilities

There exist five classes of devices attached to the MVB, which differ in their capabilities.

MVB devices shall provide a subset of the following six capabilities listed in Table 50.

Table 50 - MVB devices capabilities

Capability	Description	Class
Device_Status	the device is able to send its Device_Status when polled	1, 2, 3, 4, 5
Process_Data	the device is able to send Process Data when polled and to receive Process Data from other devices	1, 2, 3, 4, 5
Message_Data	the device is able to send Message Data when polled and to receive Message Data from other devices. This capability implies that the devices are able to execute the Real-Time Protocols and have a Network Management Agent	2, 3, 4, 5
User_Programmable	the device is able to be down-loaded with User Application programs. This capability implies the Message_Data capability.	3, 4, 5
Bus_Administrator	the device is able to act as a master. This capability implies the Message_Data, Process_Data and Device_Status capabilities and the ability to read the Device_Status of all other devices.	4, 5
TCN_Gateway	the device is able to access at least another bus (MVB or other). This capability implies the Device_Status, Process_Data and Message_Data capabilities, and the presence of a router if at least two of the busses obeys to the Real-Time Protocols.	5

3.2.2.2 Class 0 devices

Class 0 devices are not requested to support any capability.

NOTE Class 0 includes special devices, such as repeaters or star couplers, which do not participate in the application data exchange or participate in another way (for instance with another protocol).

3.2.2.3 Class 1 devices

Class 1 devices shall offer the capabilities: Device_Status and Process_Data.

NOTE In these devices, the Port_Address of the Process Data may be tied to their Device_Address, for instance, the Port_Address may be identical to the Device_Address.

3.2.2.4 Class 2 devices

Class 2 devices shall offer the capabilities: Device_Status, Process_Data and Message_Data.

NOTE Class 2 devices are intelligent field devices which can be configured over the bus, but not programmed.

3.2.2.5 Class 3 devices

Class 3 devices shall offer the capabilities: Device_Status, Process_Data, Message_Data and User_Programmable.

3.2.2.6 Class 4 devices

Class 4 devices shall offer the capabilities: Device_Status, Process_Data, Message_Data and Bus_Administrator.

NOTE The User_Programmable capability is optional.

3.2.2.7 Class 5 devices

Class 5 devices shall offer the capabilities: Device_Status, Process_Data, Message_Data and TCN_Gateway.

Class 5 devices may offer the Bus_Administrator capability.

NOTE Gateways with Bus_Administrator capability can synchronise the busses.

3.2.2.8 Device Attachment

Three different device attachment are considered:

- a) ESD: Devices equipped with the connectors carrying ESD signals according to 3.2.4;
- b) EMD: Devices equipped with the connectors carrying EMD signals according to 3.2.5;
- c) OGF: Devices equipped with the connectors carrying ESD signals according to 3.2.6.

NOTE A device may support more than one attachment.

3.2.3 Specifications common to all media

3.2.3.1 Signalling speed

The signalling speed shall be 1,5 Mbit/s $\pm$ 0,01 %, using Manchester encoding (BR = 1,5 MHz or 1,5 Mbit/s, BT = 666,7 ns).

3.2.3.2 Propagation delays

Between any two devices, the reply delay, including propagation delays and repeater delays and assuming a worst-case source delay of T_source_max (see 3.4.2.3), shall not be larger than T_reply_def = 42,7 μ s (see 3.4.2.2), except if the bus is operated in Extended Reply Mode.

If the worst case propagation delay is larger than T_reply_def, the bus may be operated in Extended Reply Delay mode, provided that the worst case round-trip delay T_reply_max is entered as a configuration parameter of all connected devices.

The difference between the maximum and the minimum propagation delay on two consecutive telegrams, considering overall propagation and repeater delays, shall at no place exceed 4,0 BT.

The sum of the interframe spacings from one master frame to another master frame shall be larger than 9,0 BT.

NOTE 1 The default value of T_{reply_def} of 42,7 μs limits the number of repeaters to four, and the transmitted distance to 2,0 km. Using a value larger than T_{reply_def} is possible with certain precautions explained in the Extended Reply Delay mode (3.4.2.4.2.)

NOTE 2 Fluctuations in propagation delay could reduce the interframe spacing and cause frame overlap. Repeaters may increase the interframe spacing, but overrun may occur if the above requirement is exceeded.

NOTE 3 Imposing a minimum sum of interframe spacings prevents a pair of fast master and slave from overrunning slower listeners.

3.2.3.3 Transceiver interface

The transceiver interface specifies for all types of media the interface between the Bus_Controller and the transceivers within a device.

This interface can remain internal to a device, but it is recommended to make it available for testing. This interface is not covered by conformance testing.

The transceiver interface is defined as an electrical interface operating with binary signals.

The transceiver interface assumes that the medium takes two distinct levels, a HIGH level and a LOW level. The HIGH and LOW levels are defined for each media.

If exposed, the transceiver interface shall consist of the following signals:

a) TxS: Transmitter_Signal

This signal controls the level of the medium. It is "0" when the medium is at the LOW level, or a "1" when the medium is at the HIGH level;

b) TxE: Transmitter_Enable

This signal becomes "1" to enable the transmitter. This signal is not needed for optical transmission. Its timing is defined for each medium;

c) RxS: Receiver_Signal

This signal expresses the state of the medium. It is "0" when the line is at the LOW level, "1" when the line is at the HIGH level. The receiver interprets undefined levels, either as LOW or as HIGH. There is no defined level when no transmitter is active, although some media define an idle level (generally LOW).

Figure 45 illustrates the transceiver interface for all three media.

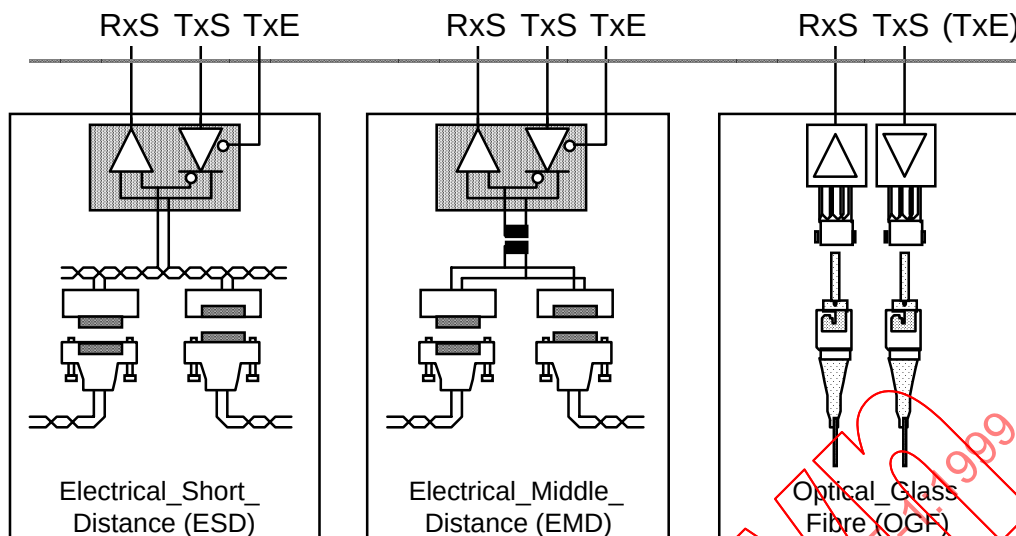


Figure 45 - Transceiver interface

3.2.3.4 Redundant medium (option)

For applications which need a high availability, this subclause defines a redundancy scheme, in which the bus may be either fully duplicated (all segments are double-line) or partially duplicated (include single-line segments and double-line segments). In case this option is used, the following specifications apply.

3.2.3.4.1 Identification

A double-line segment shall consist of two lines, operated in parallel.

All devices connected to a double-line segment shall identify the same line as Line_A, and the other line as Line_B.

3.2.3.4.2 Layout

The layout of the busses and their loading shall ensure close propagation conditions on both lines of a double-line segment, to meet 3.3.2.2.2.

The layout shall take precautions to ensure fail-independence of the redundant lines.

3.2.3.4.3 Connection between single-line and double-line segments

When a double-line segment is connected to another double-line segment by a repeater, there shall be an independent repeater for each Line_A and Line_B, and the identification of Line_A and Line_B shall be maintained over this connection.

When a double-line segment is connected to a single-line segment by one repeater, the repeater shall be connected to both lines on one side and to one line on the other, as specified in 3.3.3.2.

3.2.4 Electrical Short Distance medium (choice)

The specifications of the Electrical Short Distance medium allow segments with up to 32 devices over a distance of up to 20 m without galvanic separation. Actually, the covered distance and the number of devices are limited by the distortion caused by cables, connectors and devices, by the ground quality and by the disturbance levels, and can be increased by providing galvanic isolation.

3.2.4.1 ESD topology

The ESD medium shall consist of two conductors (with an additional equipotential conductor), terminated and biased at each end, to which the devices are attached by short stubs.

EXAMPLE - The principle connection of an ESD segment is shown in Figure 46.

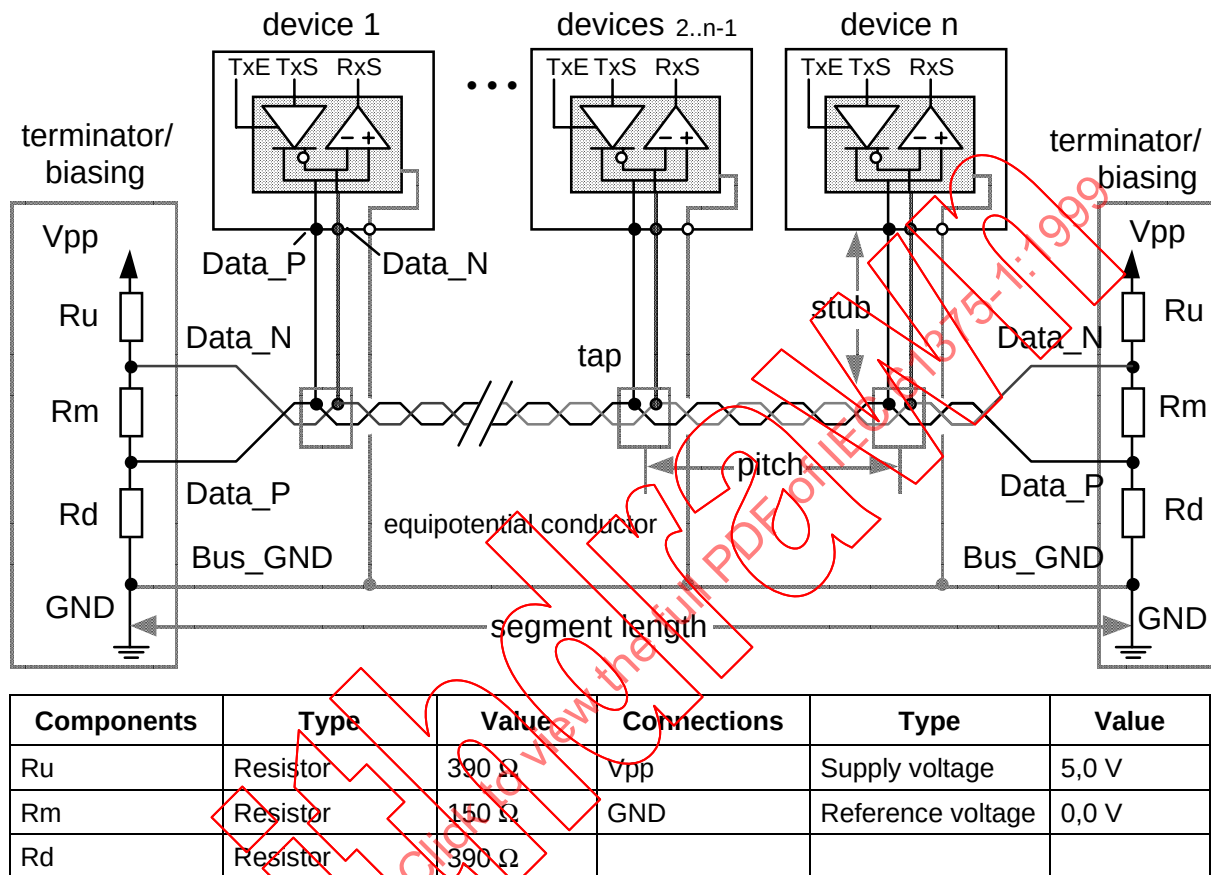


Figure 46 - Example of ESD segment

3.2.4.2 ESD configuration rules

3.2.4.2.1 ESD transmission medium

An ESD line shall consist of a pair of conductors, preferably twisted and shielded, with an equipotential conductor running in parallel.

3.2.4.2.2 ESD identification

The two conductors of a line shall be named Data_P and Data_N.

Throughout a line, the conductors Data_P and Data_N shall be marked distinctly.

The identity of the conductors shall be maintained at all connection or splicing points.

The equipotential conductor shall be called Bus_GND and marked distinctly.

3.2.4.2.3 ESD characteristic impedance

The characteristic impedance of the unloaded line shall be $120,0 \Omega \pm 10 \%$ at 1,0 BR.

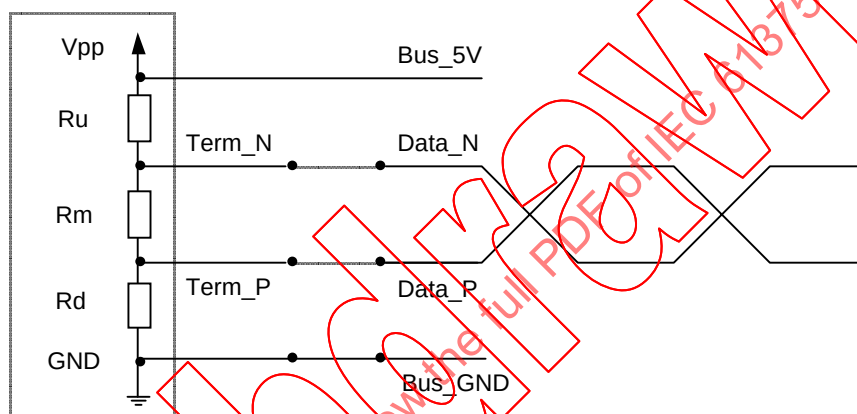
3.2.4.2.4 ESD terminator

An ESD segment shall be terminated electrically at each end by a terminator.

The terminator shall present an impedance of $120,0 \Omega \pm 10\%$ measured at 1,0 BR.

The terminator shall bias the wire pair, so that the potential of the Data_P wire is lower by at least $0,750 \text{ V} \pm 10 \%$ with respect to the Data_N wire when no transmitter is active.

EXAMPLE - The terminator shown in Figure 47 presents an impedance of $126,0 \Omega$ and biases the line with about $0,800 \text{ V}$.



Components	Type	Value	Connections	Type	Value
Ru	Resistor	390Ω	Vpp	supply voltage	$5,0 \text{ V}$
Rm	Resistor	150Ω	GND	reference voltage	$0,0 \text{ V}$
Rd	Resistor	390Ω			

Figure 47 - Example of Terminator

3.2.4.2.5 ESD attenuation due to the medium

The total voltage attenuation between any two devices located on the same line shall not exceed $8,0 \text{ dB}$.

For this test, the attenuation is measured by applying at one terminator a sinusoidal signal of differential amplitude of $4,0 \text{ Vpp} \pm 10\%$ ($1,414 \text{ Vrms}$) at 2,0 BR, and by measuring the signal at the other terminator.

3.2.4.2.6 ESD jitter due to devices, connectors and cabling

A line, with all devices and connectors in place, terminated by its characteristic impedance shall add no more than $\pm 0,1 \text{ BT}$ of edge jitter, referenced to the idealised zero-crossings.

For this test, jitter is measured by driving the line at one terminator by a source of differential amplitude $4,0 \text{ Vpp} \pm 10\%$, centred on $0,0 \text{ V}$, through a source impedance of $22,0 \Omega \pm 10\%$ and generating as signal a pseudo-random sequence of "0" and "1" Manchester symbols with a repetition period of at least 511 bits, the signal being measured at the other terminator.

NOTE 1 Interference and reflection due to impedance mismatches between the line sections, stubs, connectors or load clustering can introduce jitter in the timing of the zero-crossings.

NOTE 2 This test method is specified in ISO/IEC 8802-3.

NOTE 3 It is a good practice to cluster no more than 4 devices over a cable length of 10,0 m and to include at least 10,0 m of cable from such a cluster to the next cluster.

3.2.4.3 ESD section specifications

The ESD specifications apply to two kinds of sections:

- a) The backplane section is used in enclosures, such as racks and cabinets, where no galvanic separation is required, over distances of up to 20,0 m.
- b) The cable section is used to connect devices outside of enclosures, with or without galvanic separation. In case galvanic separation is used, distances of up to 200,0 m can be covered.

3.2.4.3.1 ESD backplane section

If a backplane bus is used as a medium, the following specifications apply.

The maximum extension of a stub shall be 10,0 cm, measured from the track to the transceiver input.

The pitch between adjacent taps shall not be less than 2,0 cm to avoid load clustering.

3.2.4.3.2 ESD cable section

If a cable section is used as medium, the specifications of the EMD cable shall apply.

In addition, the following specifications hold:

- a) a cable used for ESD shall carry at least one equipotential wire per line pair;
- b) the equipotential wire of an ESD cable shall be identified as A.Bus_GND (for Line_A) and B.Bus_GND (for Line_B, in case two redundant lines are used).

NOTE The cross-sectional area of the equipotential wire may be different from that of the data wire pair.

3.2.4.4 ESD shielding

The shield of the cable shall be connected to the casing of each connector, which shall be of conductive material.

The casing of the connector shall make electrical contact with the receptacle of the device when inserted.

The shield of the cable shall not be used in place of Bus_GND.

3.2.4.5 ESD medium-dependent interface

NOTE The Medium-dependent interface is described for a double-line segment, even when this option is not used.

3.2.4.5.1 ESD attachment

A device shall be attached to Line_A and/or Line_B through independent connection points, identified as shown in Figure 48 as:

- A.Data_P and A.Data_N; respectively
- B.Data_P and B.Data_N.

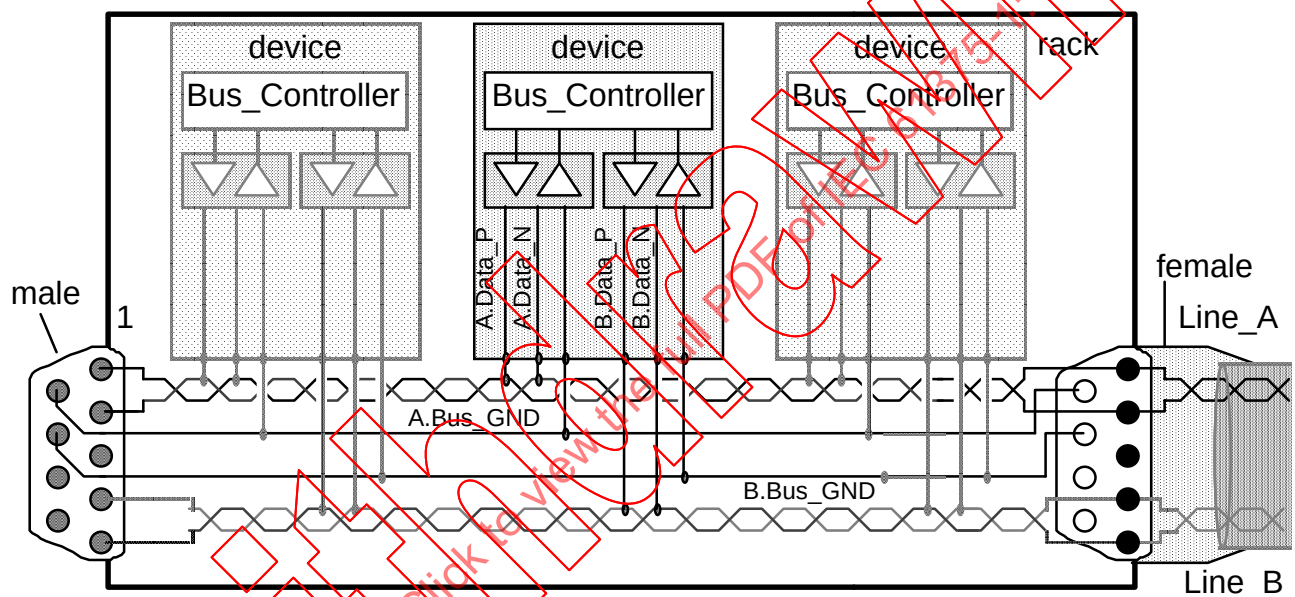


Figure 48 - ESD backplane section (double-line)

NOTE 1 This scheme encompasses a redundancy scheme in which the two redundant lines use independent twisted pairs sharing the same cable and connector. This scheme offers protection against wire, pin contact and transceiver failure, but not against cable rupture or connector removal. Field experience shows that the risks of the latter are minor.

NOTE 2 In applications where separate connectors and cables are required, the connector pin-out should be the same as for the two-connector attachment and it should have the same polarity (male or female) as a single ligne section.

3.2.4.5.2 ESD connector

Where interchangeability is required, devices shall be attached to the cable as follows:

- each device shall be inserted into the cable by two 9-pin Sub-D 9 connectors using metric screws (IEC 60807), called Connector_1 and Connector_2;
- the connector shall have a shielded, conductive casing connected to the cable shield and which makes an electrical contact with the receptacle when fastened;
- it shall be possible to connect and fasten cable connectors together, so as to provide cable and shield continuity;
- the connector (male or female) shall have the pin assignment indicated in Table 51;

- e) the connector receptables shall be marked with “MVB-S1” et “MVB-S2” to identify the ESD medium;
- f) a cable section or a backplane section shall bear a male connector at one extremity and a female connector at the other extremity;
- g) the connectors shall have the polarity and arrangement shown in Figure 49;
- h) Connector_1 shall use the male connector on the device and the female on the cable;
- i) Connector_2 shall use the female connector on the device and the male on the cable;
- j) a device or a rack shall provide through the connector for each line a power supply of $5,0\text{ V} \pm 5\%$ with a minimum supply capability of 70,0 mA and limited to 300,0 mA, whose ground is connected to the equipotential conductor;

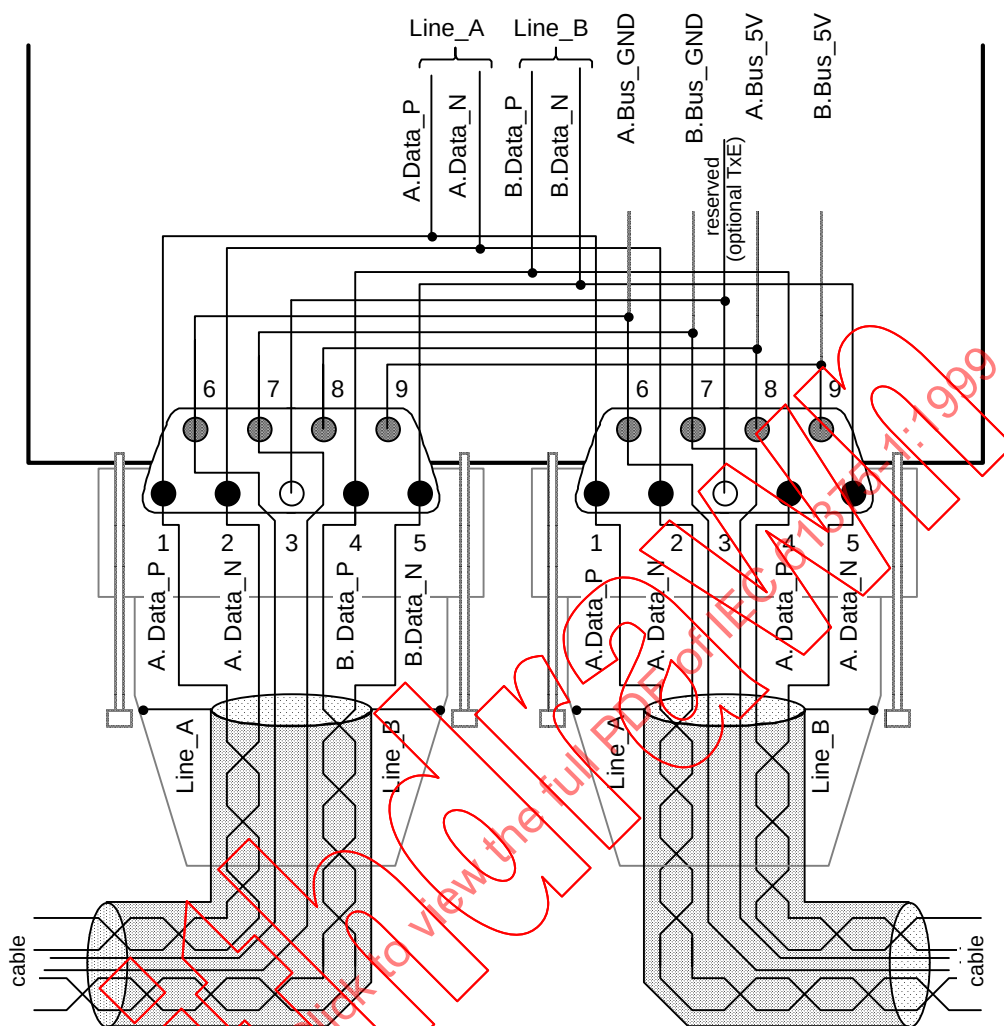


Figure 49 - ESD connector arrangement

Table 51- Pin assignment for the ESD connector

1	A.Data_P, positive wire Line_A	6	A.Bus_GND, ground Line_A
2	A.Data_N, negative wire Line_A	7	B.Bus_GND, ground Line_B
3	TxE, see 3.2.3.3 (optional)	8	A.Bus_5V, positive supply Line_A
4	B.Data_P, positive wire Line_B	9	B.Bus_5V, positive supply Line_B
5	B.Data_N, negative wire Line_B		

NOTE 1 It is recommended to arrange the two connectors side by side on the front plate of a device, Connector_1 on the left and Connector_2 on the right, to thread the wiring neatly.

NOTE 2 It may be necessary to ensure that the connector case is isolated from the device case if connection of the device cases is not advisable, for instance when large stray currents are expected.

3.2.4.5.3 ESD connector of the Terminator

The terminator shall be included into a connector, to be plugged into the empty receptacle of a device located at the end of the segment, according to the wiring shown in Figure 50:

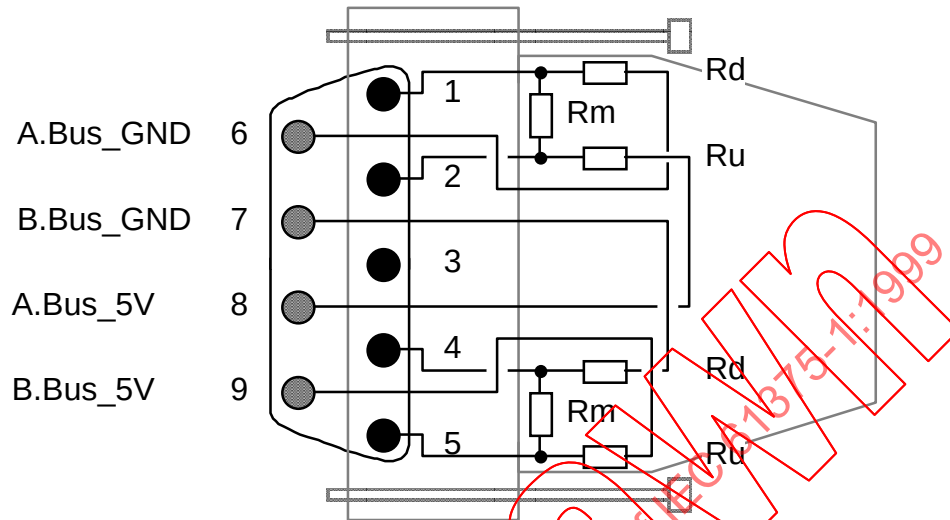


Figure 50 - ESD terminator connector arrangement

The connector containing the terminator for ESD shall be marked with a “MVB-S1”, respectively “MVB-S2”.

NOTE There are two kinds of terminator connector, male and female. The connectors for terminating ESD and EMD are distinct.

3.2.4.6 ESD Line Unit specifications

3.2.4.6.1 ESD conventions (guideline)

The characteristics of each device are measured at the points where the line is attached to the device, Data_P, Data_N and Bus_GND, as shown in Figure 48.

When measuring a transmitter, the circuit of the receiver is the normal receiving state. When measuring a receiver, the circuit of its transmitter is in a high impedance state.

If the device is attached through connectors, these are included into the measurement.

3.2.4.6.2 ESD insertion losses of a device

The load presented by a device shall conform to ISO/IEC 8482 (RS-485).

NOTE ISO/IEC 8482 specifies this value as 12 kΩ at 1,0 BR.

3.2.4.7 ESD signal wave form

The medium shall take two distinct levels:

- a HIGH level in which voltage U_p on Data_P is greater than voltage U_n on Data_N, which shall correspond to a HIGH level of the TxS or RxS signals;
- a LOW level in which voltage U_p on Data_P is smaller than voltage U_n on Data_N, which shall correspond to a LOW level of the TxS or RxS signals.

Biasing shall ensure that the line takes the LOW level when it is not driven.

EXAMPLE - The start of a frame as seen at the transmitter (TxS and TxE), on the line ($U_p - U_n$) and at the receiver (RxS), without considering timing delays, is shown in Figure 51.

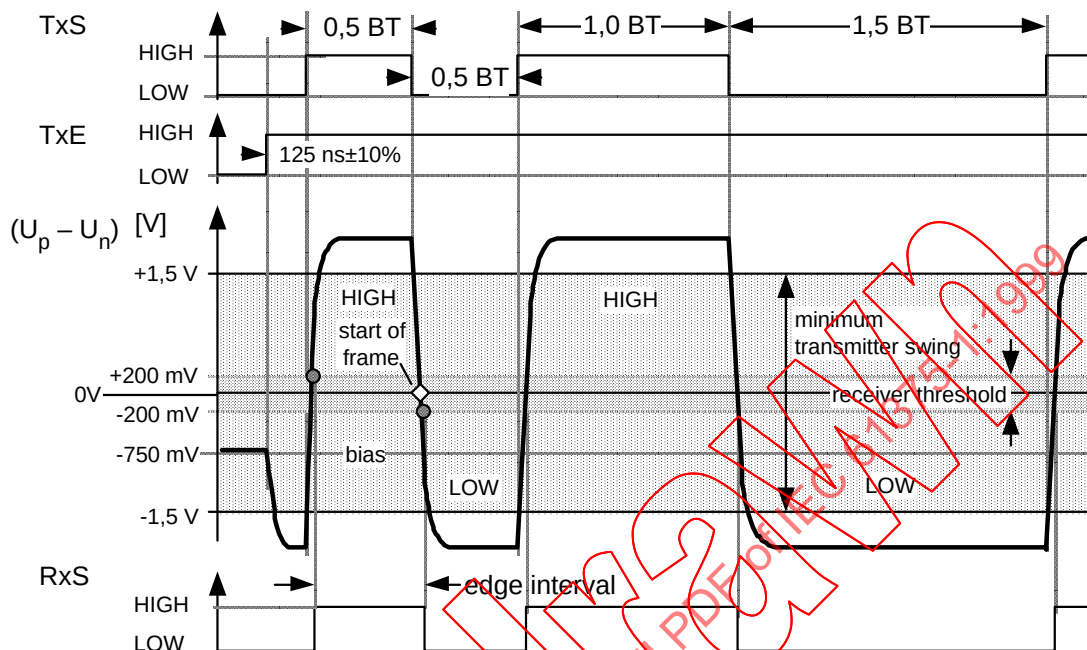


Figure 51 - Example of start of frame (ESD)

NOTE The first HIGH-to-LOW zero-crossing in the frame defines the start of the frame.

3.2.4.8 ESD transmitter

3.2.4.8.1 ESD transmitter signal

A transmitter shall conform to ISO/IEC 8482 (RS-485), with the following restrictions:

- The rise time of the signal (10% - 90%) shall be less than 0,03 BT (20,0 ns at 1,5 Mb/s), when driving a load of 54,0 Ω in parallel with 50,0 pF.
- The transmitter shall provide a low impedance differential voltage source with two active levels:
 - HIGH, when the voltage difference ($U_p - U_n$) is within:
 - $+1,5 \text{ V} < (U_p - U_n) < +5,0 \text{ V}$ when driving a resistive load of 54,0 Ω , and
 - $+1,5 \text{ V} < (U_p - U_n) < +6,0 \text{ V}$ when unloaded;
 - LOW, when the voltage difference ($U_p - U_n$) is within:
 - $-1,5 \text{ V} > (U_p - U_n) > -5,0 \text{ V}$ when driving a resistive load of 54,0 Ω , and
 - $-1,5 \text{ V} > (U_p - U_n) > -6,0 \text{ V}$ when unloaded;

NOTE Since this specification is stricter than ISO/IEC 8482, care should be taken in selecting commercial transceivers. This specification is fulfilled by transmitters which conform to IEC 61158-2.

3.2.4.8.2 ESD transmitter jitter

When sending a frame, the jitter between two consecutive edges between Start_Bit and End_Delimiter shall not exceed 10,0 ns.

3.2.4.8.3 ESD transmitter start of frame

Before sending the Start_Bit, the transmitter shall drive the line actively towards the LOW state for at least $0,125 \mu\text{s} \pm 0,010 \mu\text{s}$, as shown in Figure 51.

NOTE Driving the line LOW before the Start_Bit is good practice to achieve a sharper starting edge at the receiver.

3.2.4.8.4 ESD transmitter end of frame

After sending the last bit of a frame, the transmitter shall drive the line actively towards the LOW state for at least $0,125 \mu\text{s}$ and at most 1,0 BT, as shown in Figure 52.

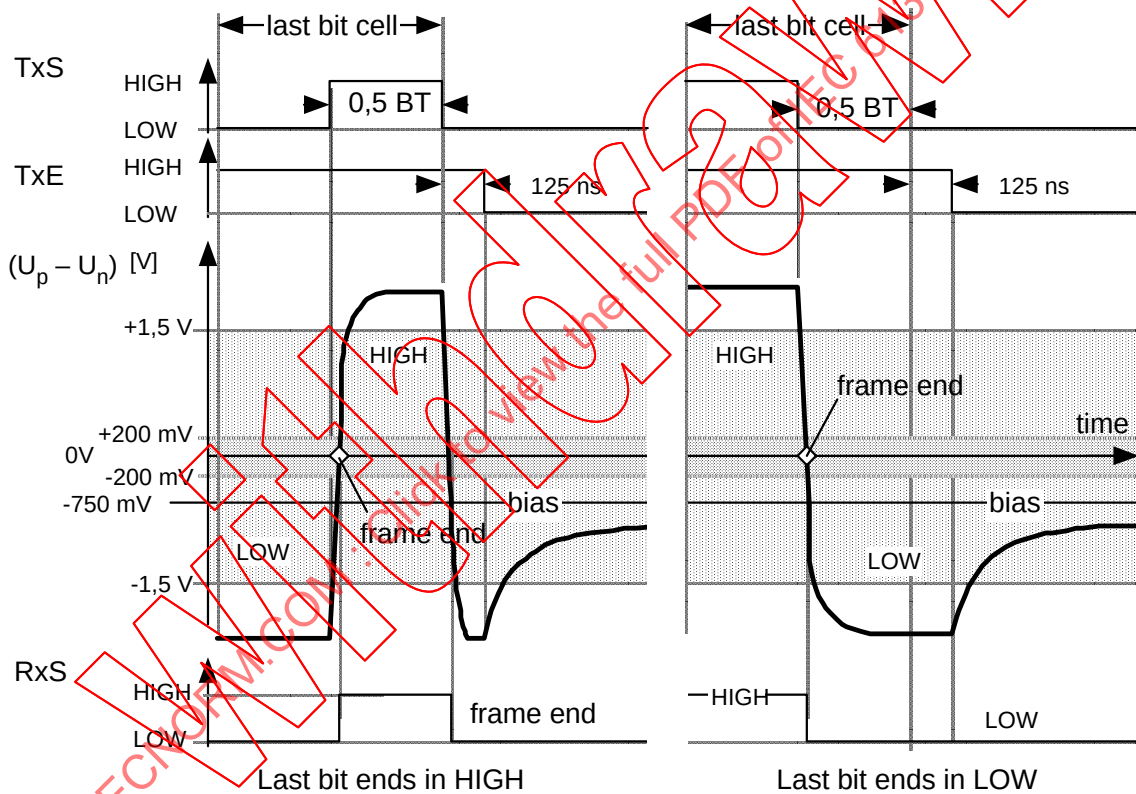


Figure 52 - End of an ESD frame (both cases)

3.2.4.9 ESD receiver

A receiver shall conform to ISO/IEC 8482 (RS-485), with the following restrictions:

- a) Depending on the differential line voltage, a receiver shall generate two levels on its RxS output:
 - a HIGH level if the voltage difference ($U_p - U_n$) is higher than +0,200 V, when the line is driven toward the HIGH level;
 - a LOW level if the voltage difference ($U_p - U_n$) is lower than -0,200 V, when the line is driven toward the LOW level or when the line is not driven and only the bias voltage appears;

- b) The receiver shall have hysteresis of at least 0,050 V, but of no more than 0,200 V;
- c) The receiver shall operate correctly in the presence of the common mode voltage specified in RS-485 with respect to the Bus_GND line.

3.2.5 Electrical Middle Distance medium (choice)

The specifications of the Electrical Middle Distance medium allow segments with up to 32 devices over a distance of up to 200 m when using the specified medium. The actual distance and the number of devices is limited by the distortion caused by cables, connectors and devices, by the ground quality and by the disturbance levels. Although the EMD has been designed for transformer-coupling, other attachments are possible.

3.2.5.1 EMD topology

The EMD medium shall consist of two conductors, chained from device to device, and terminated at each end, as shown in Figure 53.

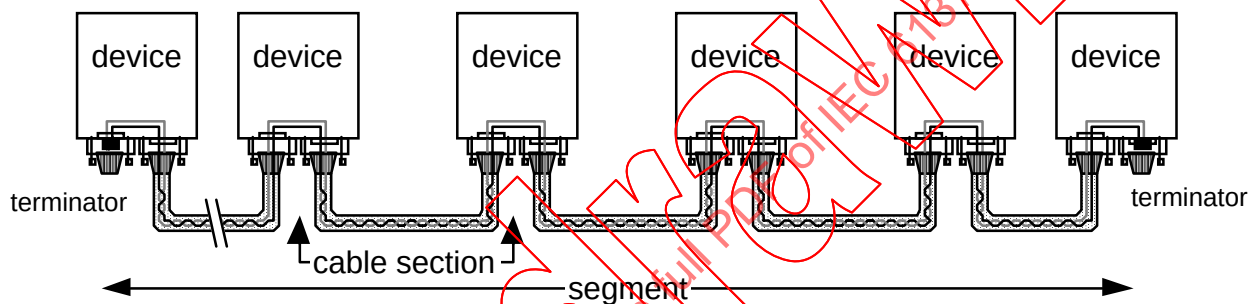


Figure 53 – EMD medium

3.2.5.2 EMD configuration rules

3.2.5.2.1 EMD transmission medium

The EMD medium shall be a shielded, twisted pair of wires.

The transmitted signal shall be a differential, polarity-sensitive voltage between these wires.

3.2.5.2.2 EMD identification

Throughout a segment, the wires shall be marked distinctly.

The identity of the wires shall be maintained at all connection or splicing points.

3.2.5.2.3 EMD attenuation due to the medium

The total signal attenuation over the medium between any two devices, expressed as the ratio of the voltage at the transmitting device to the voltage at the receiving device, shall not exceed 8,0 dB, measured at a frequency between 0,5 BT and 2,0 BT.

NOTE Attenuation limits the length of a segment for a given number of devices. With a cable attenuation of 15,0 dB/km and an insertion loss of 0,15 dB/device at 1,5 MHz, a distance of 200 m can be covered.

3.2.5.2.4 EMD jitter due to devices, connectors and cabling

A line, with all devices and connectors in place, terminated by its characteristic impedance shall add no more than $\pm 0,1$ BT of edge jitter, referenced to the idealised zero-crossings.

For this test, the jitter is measured by driving the line at one terminator by a source of differential amplitude $4,0 V_{pp} \pm 10\%$, centred on $0,0 V$, through a source impedance of $22,0 \Omega \pm 10\%$ and generating as signal a pseudo-random sequence of "0" and "1" Manchester symbols with a repetition period of at least 511 bits, the signal being measured at the other terminator.

NOTE 1 Interference and reflection due to impedance mismatches between the line sections, stubs, connectors or load clustering can introduce jitter in the timing of the zero-crossings.

NOTE 2 This test method is specified in ISO/IEC 8802-3.

NOTE 3 It is a good practice to cluster no more than 4 devices over a cable length of 10,0 m and to include at least 10,0 m of cable from such a cluster to the next cluster.

3.2.5.3 EMD terminator

A line shall be terminated at each end by a terminator presenting an impedance of $Z_w = 120,0 \Omega \pm 10\%$ and a phase angle of less than $0,087$ rad over the frequency range of $0,5$ BR to $2,0$ BR.

3.2.5.4 Cable section

The cable specifications apply to both ESD and EMD, allowing to use only one type of cable. The particularities of the ESD cable are mentioned in 3.2.4.3.2.

3.2.5.4.1 Mechanical

All sections shall consist of a shielded, jacketed cable carrying at least one twisted wire pair.

The cable shall have no less than 12 twists per metre.

NOTE A cross-sectional area of each wire between $0,34 \text{ mm}^2$ (AWG 22) and $0,56 \text{ mm}^2$ (AWG 20) is recommended.

3.2.5.4.2 Marking

The individual wires of the twisted pair shall be identified:

- a) for Line_A (even if only one pair is used) as: A.Data_P and A.Data_N;
- b) for Line_B (in case two line pairs are used) as: B.Data_P and B.Data_N.

The individual wires of the cable shall be marked distinctly.

Where the cable carries two lines, the redundant twisted pairs shall be marked distinctly.

NOTE The two pairs of wires may be geometrically laid out as a quadruple, in this case, diagonal wires shall form one pair.

3.2.5.4.3 Characteristic impedance

The bus sections shall present a differential characteristic impedance of $Z_w = 120,0 \Omega (\pm 10\%)$ to the data wires, measured with a sinusoidal signal at a frequency between $0,5$ BR and $2,0$ BR.

3.2.5.4.4 Cable attenuation

The cable shall attenuate a sinusoidal signal sent between the data wires by less than 15,0 dB/km at 1,0 BR, and by less than 20,0 dB/km at 2,0 BR.

3.2.5.4.5 Distributed capacitance

The differential (wire-to-wire) distributed capacitance of the cable shall not exceed 46,0 pF/m at 1,0 BR.

3.2.5.4.6 Capacitive unbalance to shield

The capacitive unbalance to shield shall not exceed 1,5 pF/m at 1,0 BR.

3.2.5.4.7 Crosstalk rejection

Where two pairs of wires are carried in the same cable, the signal rejection from one redundant pair to the other shall be superior to 45,0 dB in the range from 0,5 BR to 2,0 BR.

NOTE Cable for double-line segments may carry two wire pairs in the same cable.

3.2.5.4.8 Shield quality

The transfer impedance of the cable, shall, at 20,0 MHz, be less than 0,020 Ω /m, measured as specified in IEC 60096-1.

The differential transfer impedance of the cable shall be less than 0,002 Ω /m.

NOTE IEC 60096-1 mainly addresses coax cables, but it may be used for twisted pairs considering the short-circuited line pair as the inner conductor and the shield as the outer conductor.

3.2.5.4.9 Cable connector quality

All cable connections shall provide continuity of wires and shielding, with a resistance of less than 0,010 Ω .

The transfer impedance of the connector, measured at 20,0 MHz, shall be less than 0,020 Ω between one pin and shield, respectively 0,002 Ω between two pins.

NOTE 1 These requirements do not apply to connectors between vehicles.

NOTE 2 A measurement method for the transfer impedance of the conductor is available in NF C46631, 2.4.

3.2.5.5 EMD shielding

3.2.5.5.1 EMD shield continuity

A device shall connect the shields of the two cable sections to which it is attached.

When a device is removed, it shall be possible to connect the shields together, e.g. through the connectors.

The device shall provide means to connect the shield to its device ground so that a suitable place can be found for earthing the shield, or to implement a grounding concept in which all casings are connected together, as shown in Figure 54.

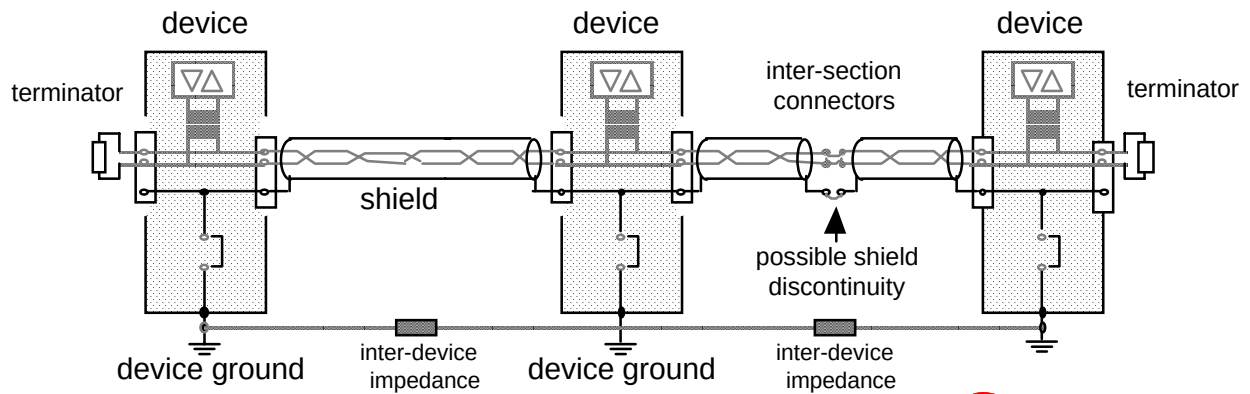


Figure 54 - Shielding (single-line segment)

NOTE The casing of a device is not necessarily connected to the receptacle of its connectors, although this is the recommended arrangement.

3.2.5.5.2 EMD shield discontinuity

In applications where large voltage differences between devices are expected, cable sections may be connected at selected places without establishing shield continuity, as shown in Figure 54.

At these places, non-standard cables, for instance using double shielding, may be used to comply with the EMC requirements.

NOTE This exception accounts for jumper cable or automatic couplers between vehicles where high stray currents are expected.

3.2.5.6 EMD medium-dependent interface

3.2.5.6.1 EMD single line attachment

A device shall be attached to the line by a passive tap.

The length of the stub, measured from the derivation point (see Figure 55) to the transceiver, shall not exceed 10,0 cm.

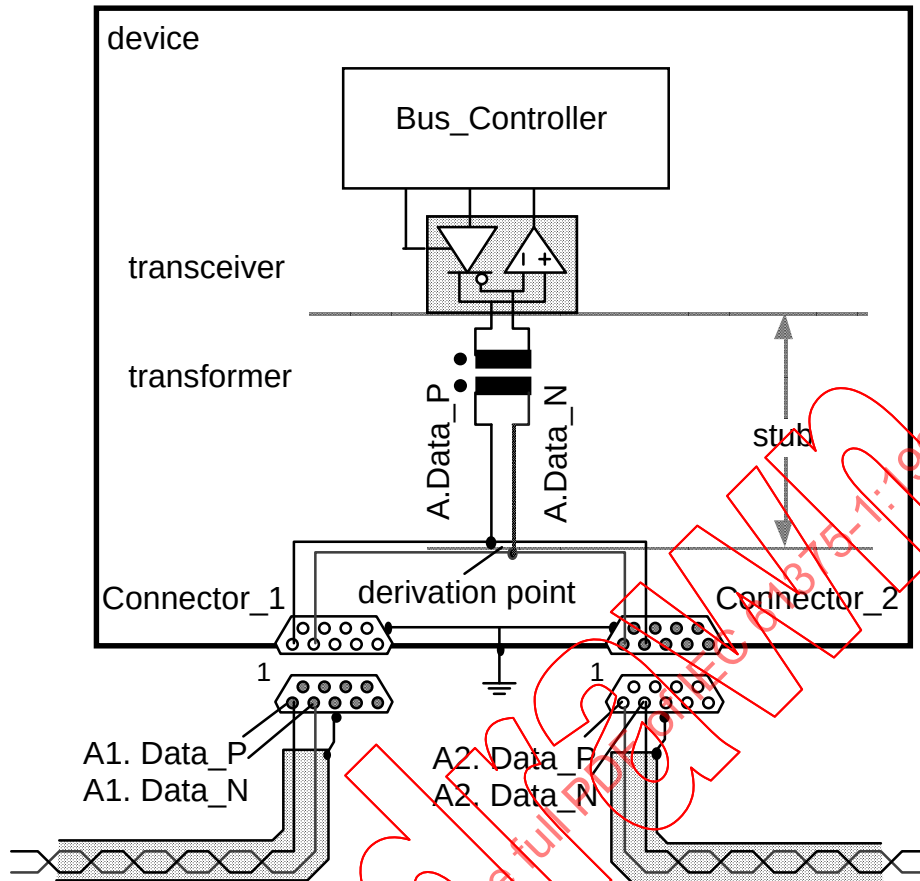


Figure 55 - Single-line device attachment

NOTE To conform to this specification, it is advisable that the device is attached by two connectors, as shown in Figure 55.

3.2.5.6.2 EMD double-line attachment (option)

A device with double-line attachment shall be capable of being attached to both a single-line or to a double-line EMD segment, as shown in Figure 56.

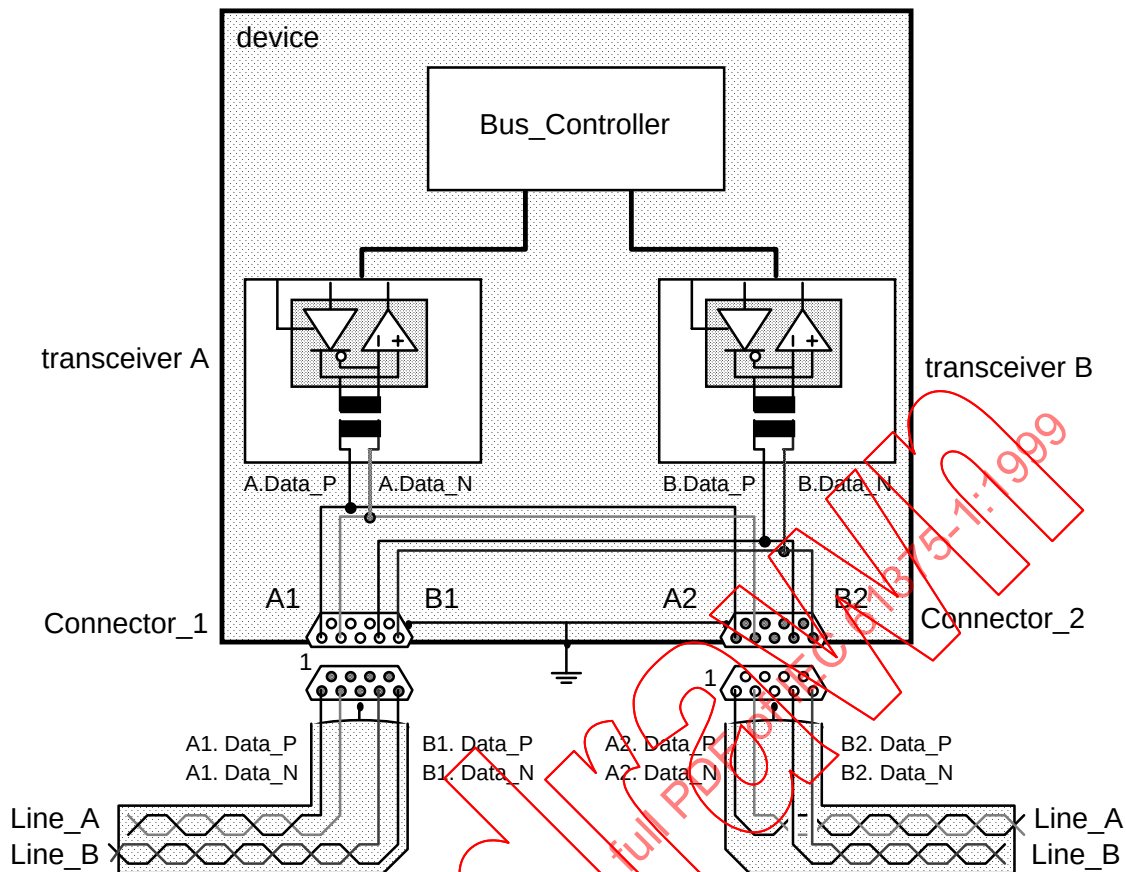


Figure 56 - Double-line device attachment to EMD

NOTE 1 This scheme encompasses a redundancy scheme in which the two redundant lines use independent twisted pairs sharing the same cable and connector. This scheme offers protection against wire, pin contact and transceiver failure, but not against cable rupture or connector removal. Field experience shows that the risks of the latter are minor.

NOTE 2 In applications where the redundant lines should be run over separate connectors, in a four connector configuration, the connector pin-out should be the same as for the two-connector attachment and it should have the same polarity (male or female) at one extremity of a cable section.

3.2.5.6.3 EMD connector

Where interchangeability is required, devices shall be attached to the cable as follows:

- each device shall be inserted into the cable by two 9-pin Sub-D 9 connectors using metric screws (IEC 60807), called Connector_1 and Connector_2;
- the connectors shall have a shielded, conductive casing connected to the cable shield and which makes an electrical contact with the receptacle when fastened;
- it shall be possible to connect and fasten cable connectors together, so as to provide cable and shield continuity;
- the connector (male or female) shall have the pin assignment shown in Table 52.
- the connector receptacle for EMD shall be marked with "MVB-M1" (for Connector_1) and "MVB-M2" (for Connector_2) to identify the EMD medium;
- a cable section shall bear a male connector at one extremity and a female connector at the other extremity;
- the connectors shall have the following polarity and arrangement, as shown in Figure 57;

- h) Connector_1 shall use the male connector on the device and the female on the cable;
- i) Connector_2 shall use the female connector on the device and the male on the cable.

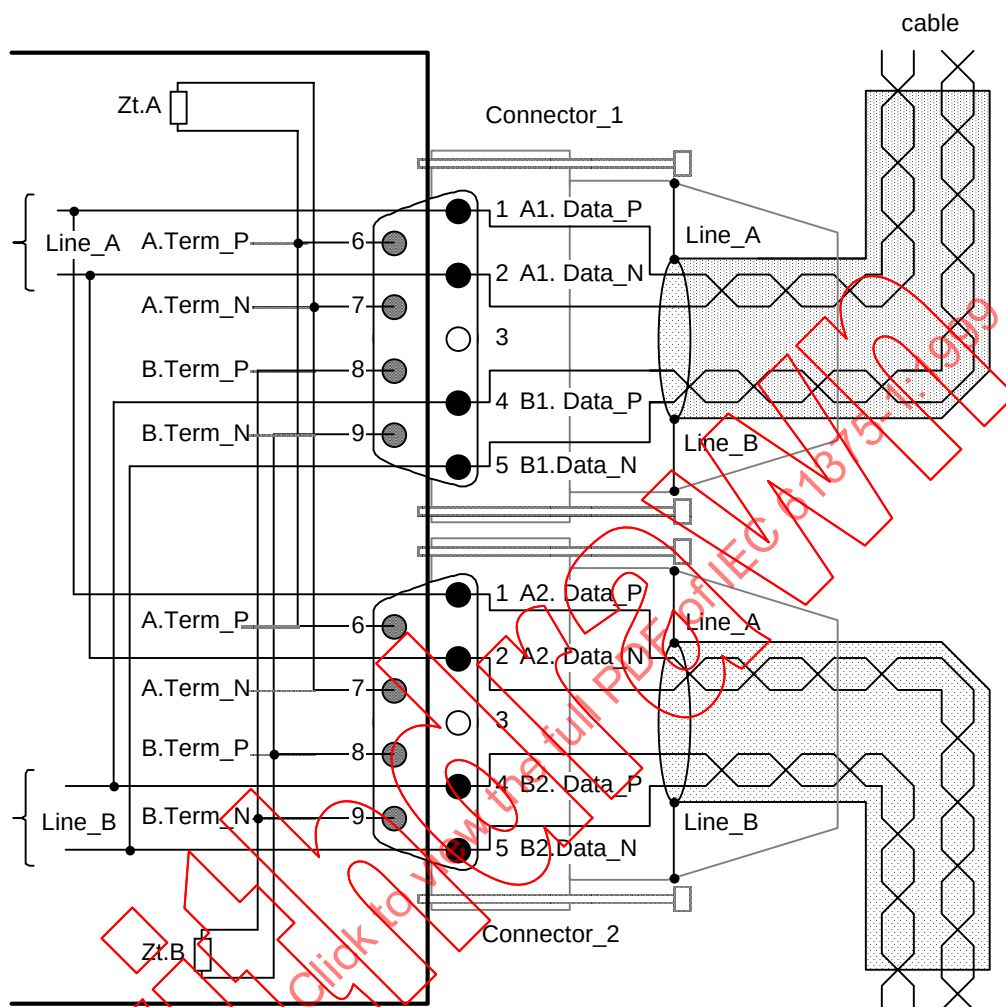


Figure 57 - EMD connectors arrangement

Table 52 - Pin assignment for the EMD connector

1	A.Data_P positive wire of Line_A	6	A.Term_P, positive pole of Terminator Line_A
2	A.Data_N negative wire of Line_A	7	A.Term_N, negative pole of Terminator Line_A
3	reserved for TxE. see 3.3.2.3.3 (option)	8	B.Term_P, positive pole of Terminator Line_B
4	B.Data_P positive wire of Line_B	9	B.Term_N, negative pole of Terminator Line_B
5	B.Data_N negative wire of Line_B		

NOTE 1 It is recommended to arrange the two connectors side by side on a device, Connector_1 on the left and Connector_2 on the right, to thread the wiring neatly.

NOTE 2 It may be necessary to ensure that the connector case is isolated from the device case if connection of the device cases is not advisable, for instance when large stray currents are expected.

3.2.5.6.4 EMD terminator strapping

To connect the terminator in the extremity device of a segment, the connector shall be strapped by connecting pin 1 to pin 6, pin 2 to 7, pin 4 to pin 8, and pin 5 to pin 9, as shown in Figure 58.

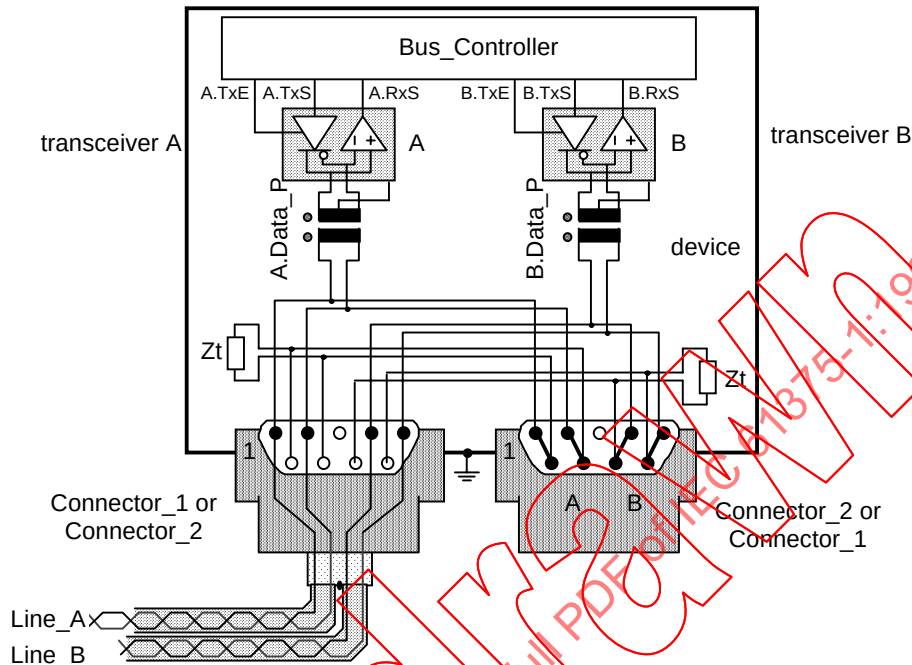


Figure 58 - EMD terminator strapping

The terminator connectors for EMD shall be marked with "MVB-M1" (to be plugged as Connector_1) and "MVB-M2" (to be plugged as Connector_2).

3.2.5.7 EMD Line_Unit specifications

Even when only Line_Unit A is mentioned, this subclause also applies to Line_Unit B.

3.2.5.7.1 EMD Line_Unit Test Circuit (guideline)

The characteristics of a device are measured at the connection points, where the cable is attached to the device, possibly through a connector, indicated in Figure 56 as:

- A1.Data_P, A1.Data_N and
- A2.Data_P, A2.Data_N.

The output or the input signal of a device is the differential voltage ($U_p - U_n$) at the connection point.

For conformity testing, it is not necessary to access the Line_Unit interface, as shown in Figure 59. However, timing diagrams refer to the Line_Unit interface signals, especially TxS and RxS, to explain the specification.

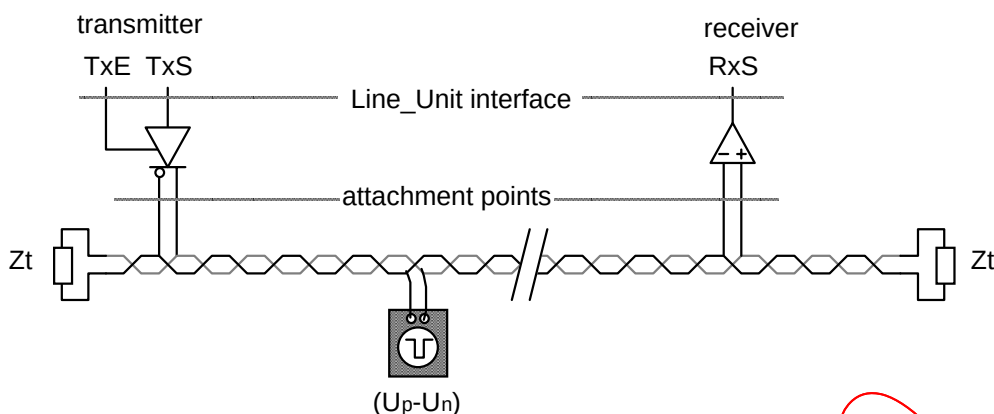


Figure 59 - Measurement of an EMD device

When measuring a transmitter, the circuit of the receiver is in the normal receiving state. When measuring a receiver, the circuit of its transmitter is in a high impedance state.

Insertion loss is measured by applying the sinusoidal signal of a generator (of internal impedance = Z_t) through 20,0 m of cable to the points A1.Data_P and A1.Data_N and by measuring the signal with a voltmeter (connected in parallel with an impedance Z_t) at the extremity of another 20,0 m of cable attached to the points A2.Data_P and A2.Data_N, or vice-versa, as shown in Figure 60.

The attenuation is defined as the ratio expressed in dB of two differential voltages:

- a first voltage being set to 4,0 Vpp when the device is removed and the cable connectors coupled;
- a second voltage being measured when the device is inserted.

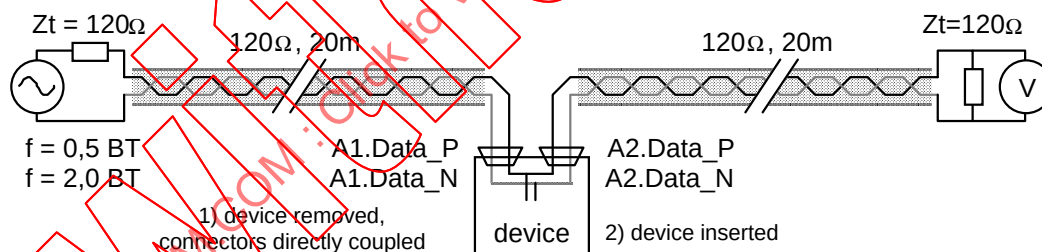


Figure 60 - Attenuation measurement

3.2.5.7.2 EMD insertion losses of a device

A device either:

- with receiver in normal operation and with transmitter in the high-impedance state; or
- with no power applied to either receiver or transmitter.

shall introduce less than 0,15 dB of attenuation at the frequencies of 0,5 BR and 2,0 BR.

3.2.5.7.3 EMD galvanic separation

Between the device ground and any of the points:

- A1.Data_P,
- A1.Data_N,

- c) A2.Data_P, and
- d) A2.Data_N

the isolation voltage and resistance shall exceed the value specified by IEC 60571.

NOTE These values are, in the present edition, 0,500 kVrms and 1,0 MΩ respectively.

3.2.5.8 EMD signal waveform

The medium shall take two distinct levels:

- a) a HIGH level when the potential difference ($U_p - U_n$) between Data_P and Data_N is positive, which shall correspond to a HIGH level of the TxS or RxS signals;
- b) a LOW level when the potential difference ($U_p - U_n$) between Data_P and Data_N is negative, which shall correspond to a LOW level of the TxS or RxS signals.

The start of the frame is defined by the first HIGH-to-LOW zero-crossing in the frame.

NOTE The state of the line is undefined when the line is not driven.

EXAMPLE - The start of a frame as seen at the transmitter (TxS), on the line ($U_p - U_n$) and at the receiver (RxS), without considering the delays on the line or in the devices is shown in Figure 61.

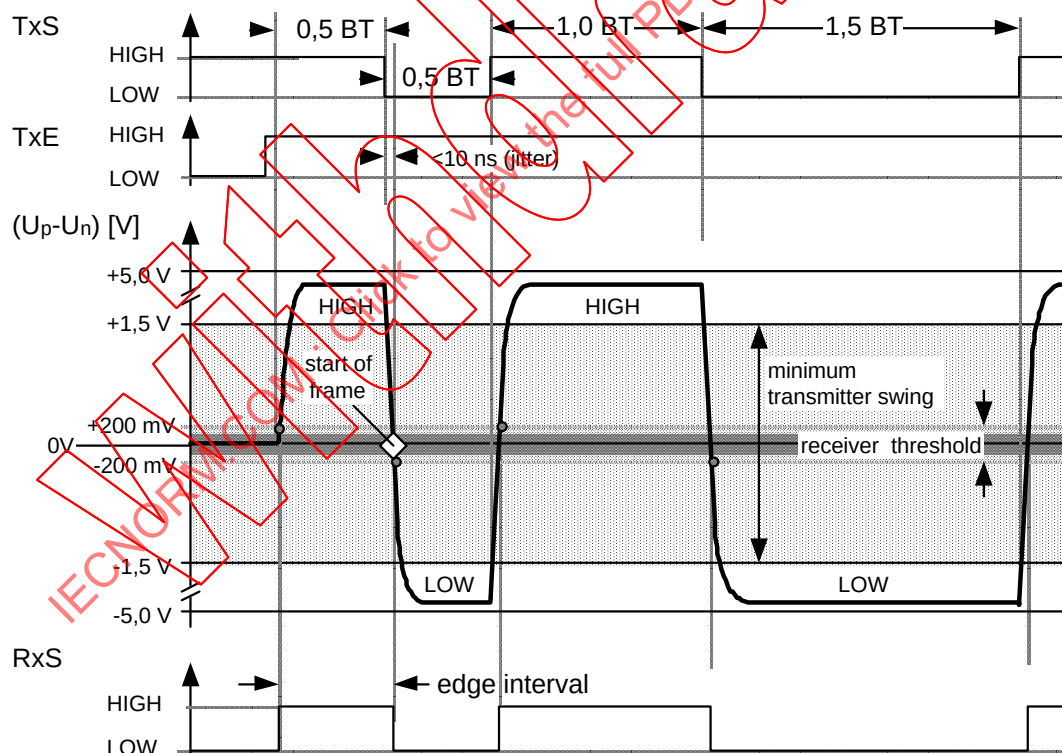


Figure 61 - Example of start of an EMD frame

3.2.5.9 EMD transmitter specifications

3.2.5.9.1 EMD transmitter test circuit (guideline)

To approximate the loading of the transmitter with a cable and devices, four test circuits are specified, corresponding to four test cases:

a) Light test circuit:

The light test circuit simulates a line where only one device is present, the value of the total resistive load is equal to $0,5 \pm 1\%$ that of the terminator;

b) Heavy test circuit:

The heavy test circuit simulates a fully loaded bus, the value of the total resistive load is equal to $0,42 \pm 1\%$ of that of the terminator;

c) Idling test circuit:

The idling test circuit simulates the behaviour of a 200,0 m cable without resistive consumers, the capacitors have a value of $150,0 \text{ pF} \pm 10\%$ each, the resistors a value of $2,2 \Omega \pm 1\%$;

d) Short test circuit:

The short test circuit simulates a line failure, it consists only of a current measurement circuit.

These circuits are shown in Figure 62.

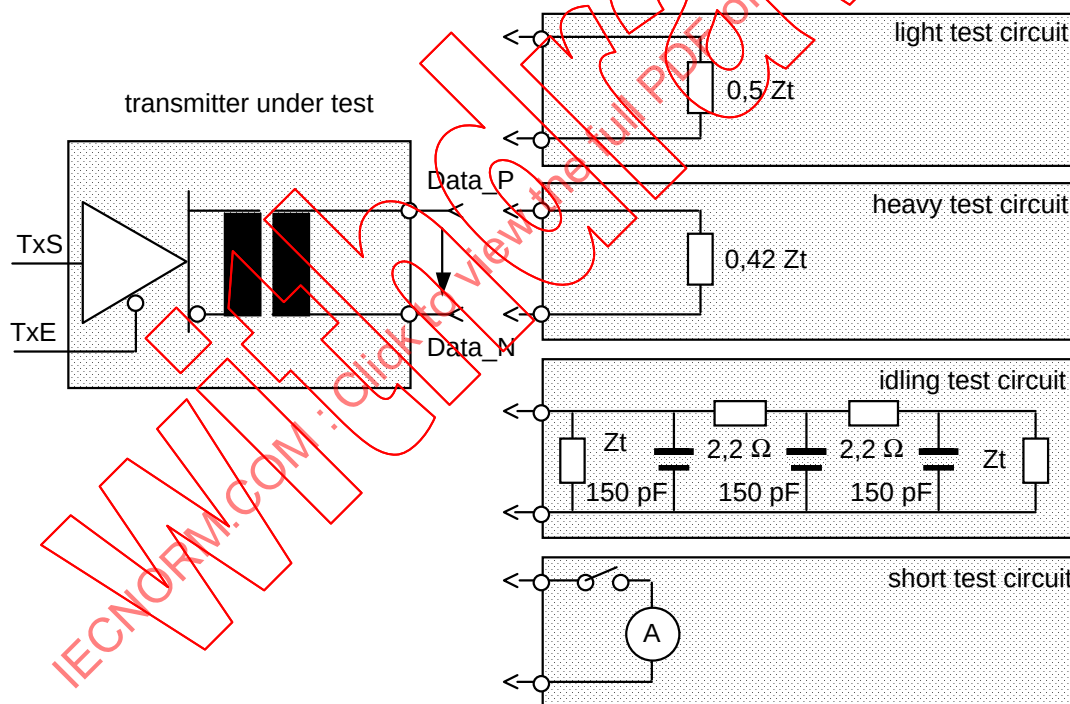


Figure 62 - EMD Transmitter test circuits

The stationary amplitude is defined as the average amplitude of the output signal when transients settle.

3.2.5.9.2 EMD transmitter output signal

The transmitter shall be a differential driver.

NOTE Due to the data encoding (3.3.1), the transmitter generates pulses which are either one bit in length (1,0 BT), a half-bit in length (0,5 BT) or one-and-a-half bit in length (1,5 BT).

These specifications apply to 0,5 BT, 1,0 BT and 1,5 BT pulses, positive or negative.

When connected to both the heavy and the light test circuit of 3.2.5.9.1, the transmitter shall comply with the following requirements:

- the output signal shall be alternately positive and negative;
- the amplitude of the output signal shall be at least $\pm 1,5$ V when connected to the heavy test circuit and at most $\pm 5,5$ V when connected to the light test circuit;
- the difference between the positive and the negative stationary amplitude in two consecutive pulses shall not exceed 0,10 V;
- the slew rate of the output signal shall be less than 0,200 V/ns at any time and more than 15 mV/ns within 0,100 μ s of the zero-crossing;
- the overshoot of the output signal, defined as the ratio of the maximum amplitude to the stationary amplitude shall not exceed 10 % of its stationary amplitude;
- the edge distortion of the output signal, defined as the time difference between the idealised and the actual zero crossing, shall not exceed $\pm 2\%$ of one Bit Time.

EXAMPLE - A typical waveform in the middle of a frame is shown in Figure 63.

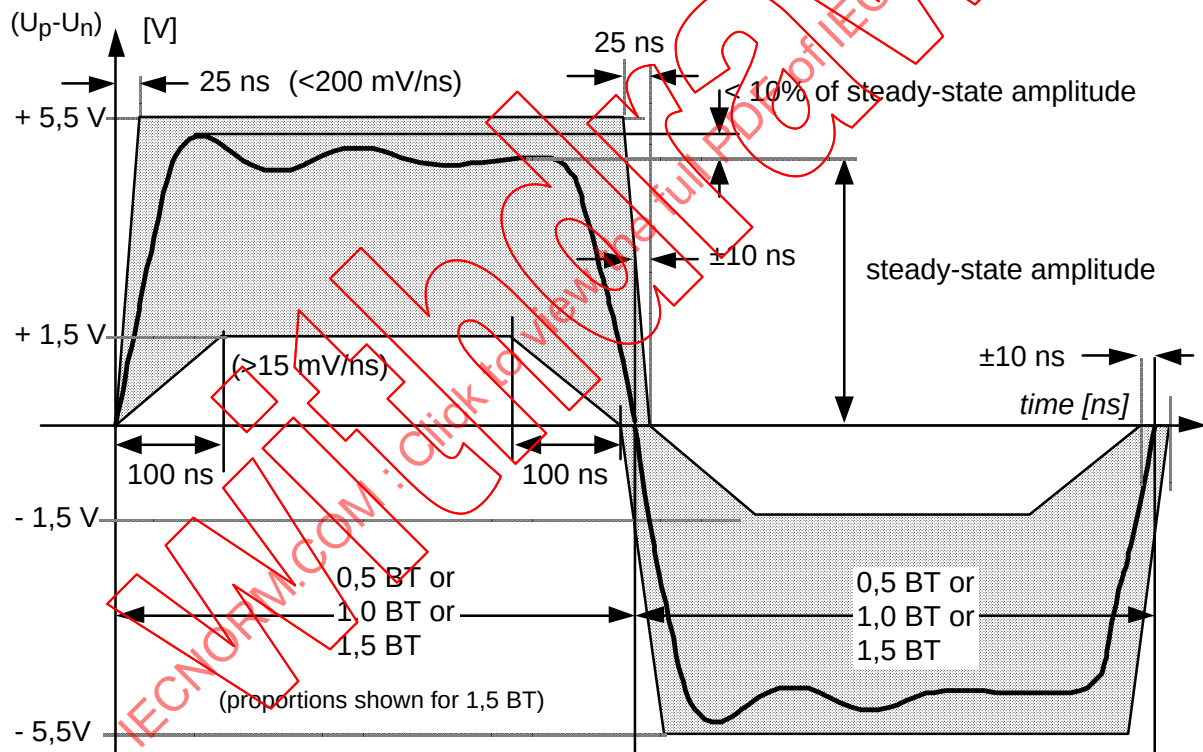


Figure 63 - Example of pulse waveform at EMD transmitter

3.2.5.9.3 EMD transmitter noise

Any noise generated by a transmitter which is not transmitting shall not exceed a value of 5,0 mVrms over the frequency range 1,0 kHz to 4,0 BR.

3.2.5.9.4 EMD transmitter end of frame

After the End Delimiter, the output signal shall reach 0,10 V within 0,30 μ s and at no time after the first zero-crossing exceed 0,20 V under the following test conditions:

- the transmitter transmits the longest possible frame;

- b) the frame data bits are a random sequence of "1" and "0";
- c) the frame is closed with the End Delimiter as specified in 3.3.1.6;
- d) the transmitter drives the idling test circuit of 3.2.5.9.1;
- e) the output signal amplitude is greater than 4,5 V before the transmitter is disabled.

EXAMPLE - The end of frame when the last data bit is a "1" is shown in Figure 64.

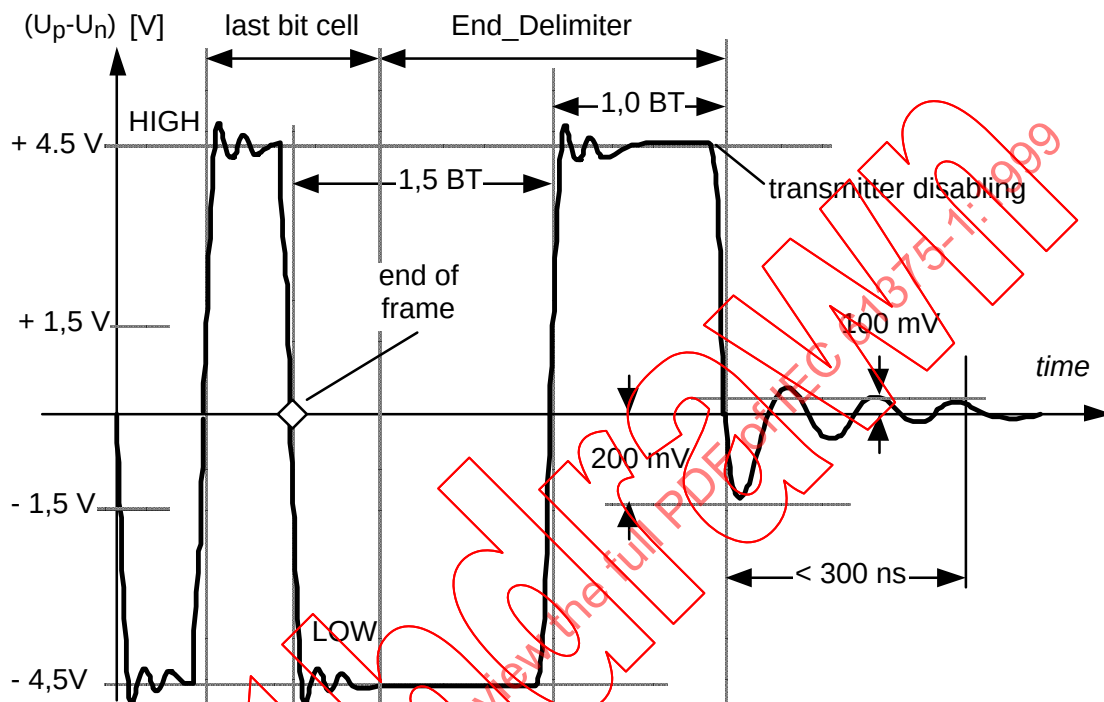


Figure 64 - Example of end of EMD frame

NOTE Line ringing after transmitter idling can be minimised by balancing the d.c.-components in each bit cell.

The last zero-crossing before the End Delimiter shall define the end of frame.

3.2.5.9.5 EMD transmitter fault tolerance

A transmitter, either when enabled and not, shall tolerate the application of the short test circuit (3.2.5.9.1) at the connection point until thermal stability is reached, and shall resume normal operation after the short test circuit is removed.

The short circuit current shall not exceed 1,0 A.

NOTE For conformance testing, thermal stability should be reached after 1 h.

3.2.5.10 EMD receiver specifications

3.2.5.10.1 EMD receiver test signal (guideline)

The characteristics of a receiver are tested by applying a test signal to the connection points and observing the frames received, either through direct access to the device, or through a device-internal response, such as response to a master frame.

To test the receiver, a test signal with the following shape, as shown in Figure 65, is applied at the connection point:

- when the amplitude of the test signal is less than 0,20 V, the slope of the test signal exceeds 4,0 mV/ns (0,20 V / 50,0 ns);
- to test sensitivity, the test signal remains above 0,30 V for a time period that starts after 0,1 μ s of the preceding zero-crossing and that lasts at least (0,5 BT - 0,2 μ s), respectively (1,0 BT - 0,2 μ s) and (1,5 BT - 0,2 μ s), whilst its peak amplitude will vary between 0,33 V and 5,00 V;
- to test insensitivity, the test signal amplitude does not exceed a prescribed value;
- to test edge distortion, the crossing of the zero voltage is varied by a prescribed value.

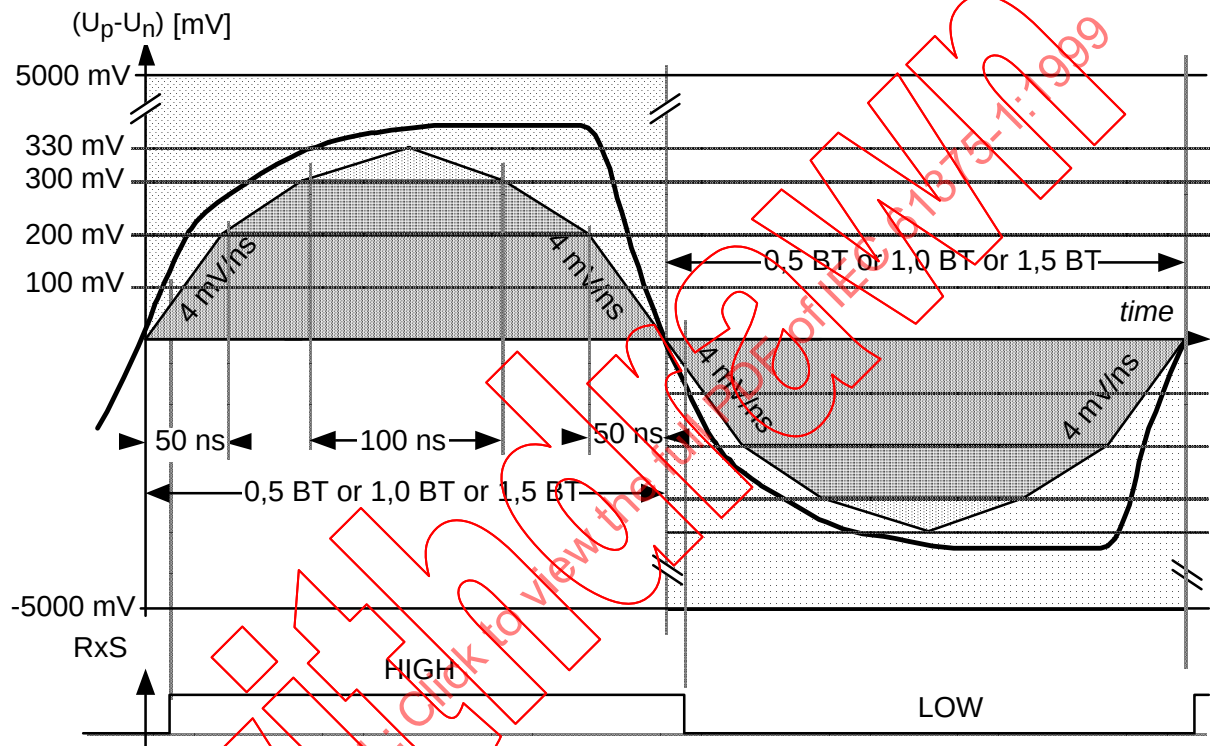


Figure 65 - EMD Receiver test signal

The receiver detects a frame error as either a silence, a collision, a not valid frame or a not correct frame (incorrect frame size, incorrect Frame Data bits or incorrect CRC).

3.2.5.10.2 EMD receiver polarity

NOTE This specification is not part of the conformance testing, since the Line_Unit interface is not necessarily exposed. It is part of the type-testing of the transceiver.

Depending on the state of the line, a receiver shall deliver a binary RxS signal with two states, HIGH and LOW, as follows:

- a HIGH level if the differential line voltage ($U_p - U_n$) exceeds +0,200 V;
- a LOW level if the differential line voltage ($U_p - U_n$) becomes less than -0,200 V.

The receiver shall have an hysteresis of at least 50 mV, but of no more than 0,25 V;

3.2.5.10.3 EMD receiver sensitivity

A receiver receiving frames containing 64 random data bits, at the rate of 1000 frames per second shall detect no more than three frame errors in $3 \times 10^{+6}$ frames, when the amplitude of the test signal defined in 3.2.5.10.1 varies between its minimum and maximum value.

3.2.5.10.4 EMD receiver insensitivity

The receiver shall not detect a valid frame when the test signal amplitude is less than 0,100 V.

NOTE This threshold may be raised in certain applications.

3.2.5.10.5 EMD receiver edge distortion

A receiver receiving frames containing 64 random data bits, at the rate of 1000 frames per second shall detect no more than three frame errors in $3 \times 10^{+6}$ frames when a test signal is applied, which edges cross the zero voltage at random within $\pm 10\%$ of 1,0 BT around the expected crossing position.

3.2.5.10.6 EMD receiver noise rejection

A receiver receiving frames containing 64 random data bits, at the rate of 1000 frames per second, and with a test signal amplitude of 0,700 V (1,400 V peak-to-peak) shall detect no more than three frame errors in $3 \times 10^{+6}$ frames when operating:

- in the presence of a common-mode sinusoidal signal (applied between the device ground and both Data_P and Data_N) with an amplitude of 4,0 Vrms in the frequency range of 65,0 Hz to 1,5 MHz or
- in the presence of an additive quasi-white Gaussian noise (applied between Data_P and Data_N) distributed over a bandwidth of 1,0 kHz to 4,0 MHz at an amplitude of 0,140 Vrms.

3.2.6 Optical Glass Fibre medium (choice)

The specifications of the Optical Glass Fibre medium allow the use of optical fibres bussed through star couplers over distances up to 2000 m when the specified medium is used. The actual distance is limited by the distortions in fibres and connectors and by the characteristics of the repeaters and star couplers.

3.2.6.1 OGF topology

The optical medium shall consist of a pair of fibres forming a full-duplex point-to-point link, as shown in Figure 66.

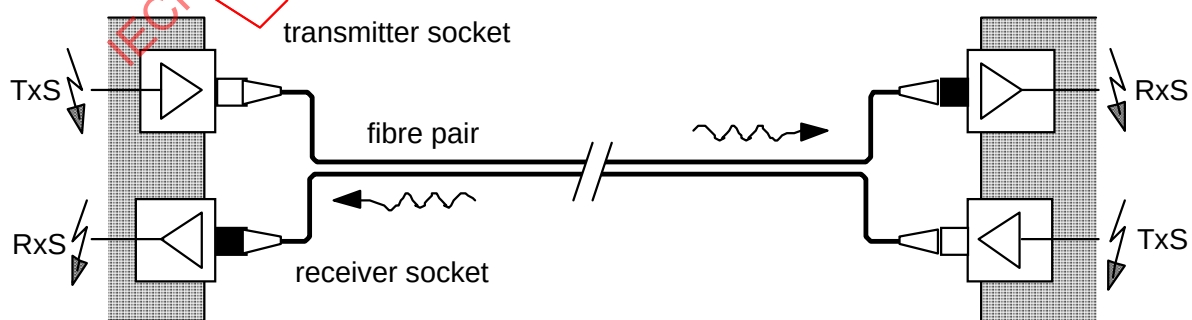


Figure 66 - Optical link

Optical fibres may be connected from device to device or may be bussed through a star coupler, preferably of the active type (3.2.6.7).

3.2.6.2 OGF optical cable and fibre

The following optical cable and fibre specification applies to a mounted cable with a connector at each end of a fibre:

- a) the fibre shall be a multiple-mode step index silica fibre with a core of 200/230 μm and a cladding eccentricity error of less than 5,0 μm ;
- b) the attenuation of a cable of 10,0 m shall be less than 1,8 dB from connector to connector, measured at a wavelength of 840,0 nm $\pm$ 30,0 nm. For longer distances, the additional attenuation shall be less than 6,0 dB /km;
- c) the minimum bending radius of a cable shall be 50,0 mm;
- d) the cable shall conform to IEC 60571 with respect to vibration test;
- e) the cable shall conform to IEC 60794-1 (E3, E6, E8) with respect to mechanical stability;
- f) the cable shall conform to IEC 60874-6 with respect to long-term stability;
- g) the cable shall conform to IEC 60332-1 regarding fire resistance;
- h) the cable shall conform to IEC 60245 Step 1 with respect to oil resistance;
- i) the cable shall conform to IEC 60304 with respect to the markings.

Two or more fibres may be located in the same cable.

When using multifibre cables, colour coding of boots is allowed.

NOTE It is recommended to identify distinctly the two extremities of a fibre, one for the transmitter side and one for the receiver side.

3.2.6.3 OGF medium-dependent interface

The connector for single fibres shall be an IEC 60874-10, (ST-bayonet) connector.

This component should preferably integrate the transmitter and the receiver as shown in Figure 67.

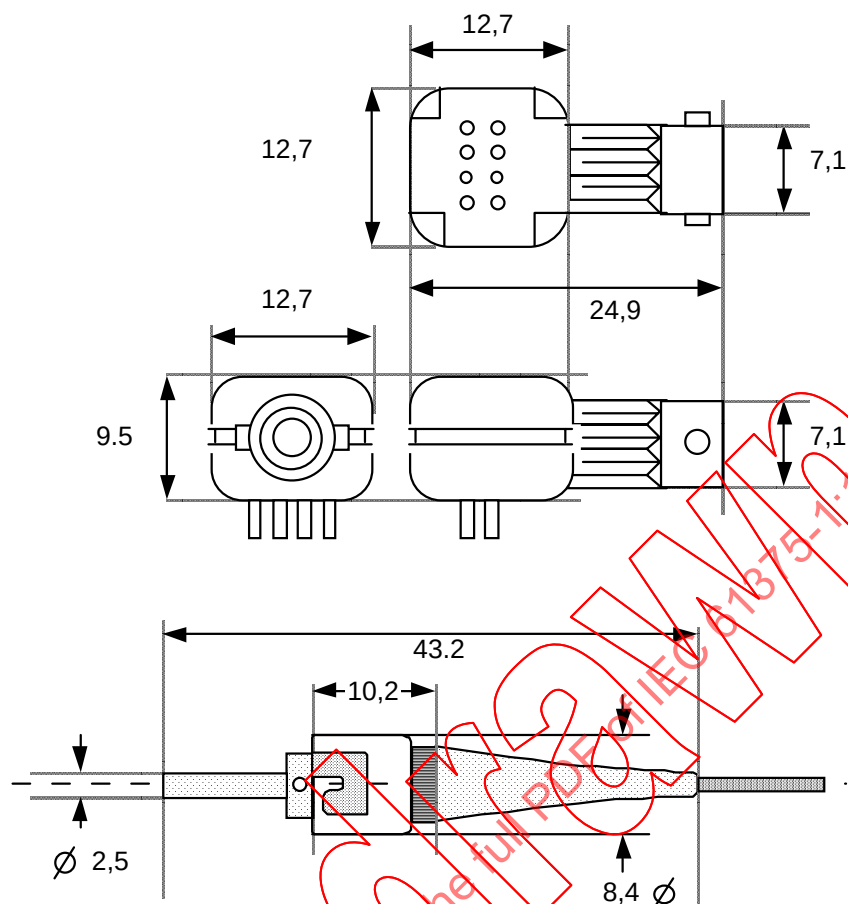


Figure 67 - Optical connector (dimensions in millimeters)

The transmitter and the receiver connector shall be adequately identified, preferably:

- light grey for the transmitter;
- dark grey for the receiver.

NOTE The components and tools of the ST-connector and fibre can be used for multiple fibre assembly.

3.2.6.4 OGF test signal (guideline)

The binary transmission uses two optical levels, the least energy level being termed LOW and the highest energy level being termed HIGH.

The LOW level corresponds to the LOW level on RxS or TxS, the HIGH level corresponds to the HIGH level on RxS or TxS.

The test signal is a well-formed frame consisting of pulses of varying amplitude with edges at a distance of 0,5 BT, 1,0 BT and 1,5 BT, with an adjustable jitter between two consecutive edges, as shown in Figure 68.

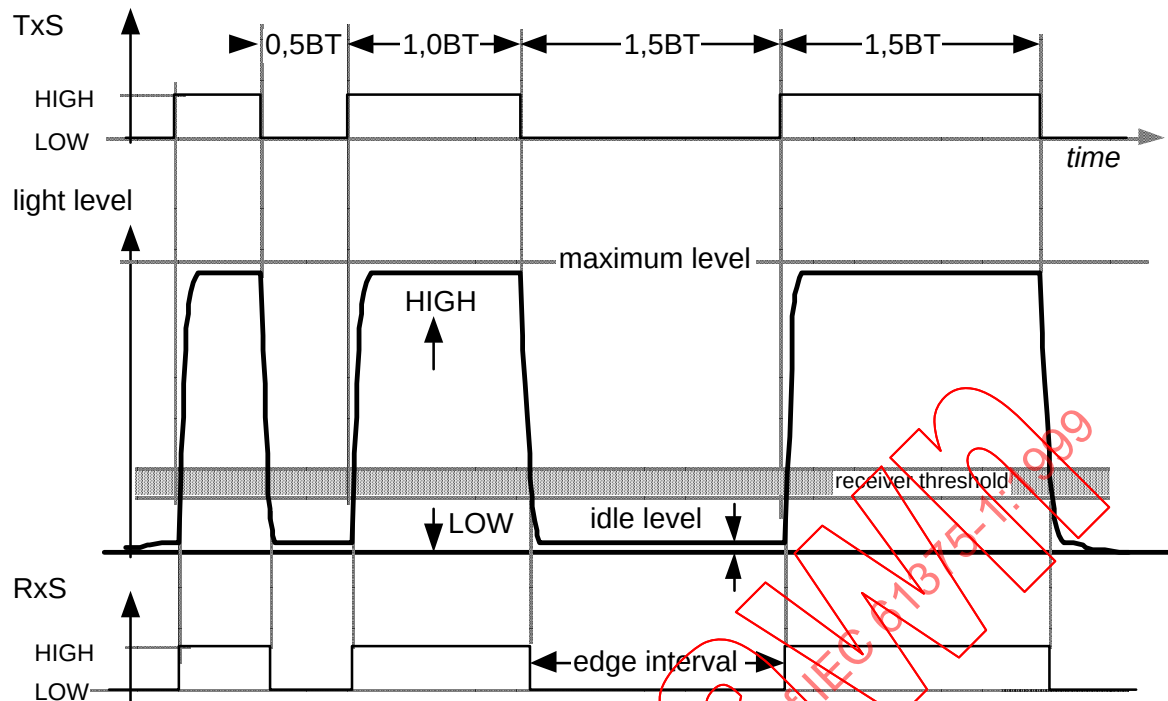


Figure 68 - Example of start of OGF frame

3.2.6.5 OGF transmitter specifications

3.2.6.5.1 OGF optical levels

The optical transmitter shall comply to the following specifications:

- the transmitter shall transmit two distinct optical levels;
- the peak emission wavelength shall be $840,0 \text{ nm} \pm 30,0 \text{ nm}$;
- the spectral width shall be smaller than $40,0 \text{ nm}$;
- the peak output power measured after $1,0 \text{ m}$ of fibre, corresponding to the HIGH state shall be comprised between $-7,5 \text{ dBm}$ and $-4,5 \text{ dBm}$;
- the minimum power, corresponding to the LOW state, shall be less than $-40,0 \text{ dBm}$;
- the rise time of the signals from LOW to HIGH (10% to 90%) shall be less than $10,0 \text{ ns}$;
- the fall time of the signals from HIGH to LOW (90% to 10%) shall be less than $10,0 \text{ ns}$;
- the difference between fall and rise time shall be less than $10,0 \text{ ns}$.

3.2.6.5.2 OGF transmitter edge distortion

The edge distortion of the output signal, defined as the time difference between the idealised and the actual transitions, shall not exceed $10,0 \text{ ns}$ over a whole frame.

3.2.6.5.3 OGF end of frame

The transmitter shall switch off after the End Delimiter, as specified in 3.3.1.6.

3.2.6.6 OGF receiver specifications

3.2.6.6.1 OGF receiver

NOTE The test of these specifications assume that the electrical side of the receiver is accessible. If the optical receiver is type-tested according to this specification, this test may be omitted in testing conformance of the device.

The receiver shall comply to the following specifications:

- the input power corresponding to a LOW signal shall be $< -40,0$ dBm ($0,1 \mu\text{W}$) at $840,0$ nm;
- the input power corresponding to a HIGH signal shall be $> -24,0$ dBm ($4,0 \mu\text{W}$) at $840,0$ nm;
- the receiver shall operate correctly up to a light level of $-10,0$ dBm ($63,0 \mu\text{W}$);
- the rise time of the output signal (LOW to HIGH) shall be less than $10,0$ ns;
- the fall time of the output signal (HIGH to LOW) shall be less than $10,0$ ns;
- the difference between fall and rise time shall be less than $10,0$ ns.

3.2.6.6.2 OGF receiver edge distortion

The receiver shall recognise a frame correctly even if the next edge appears $0,125 \mu\text{s}$ too late or too early with respect to the expected edge position, as shown in Figure 69.

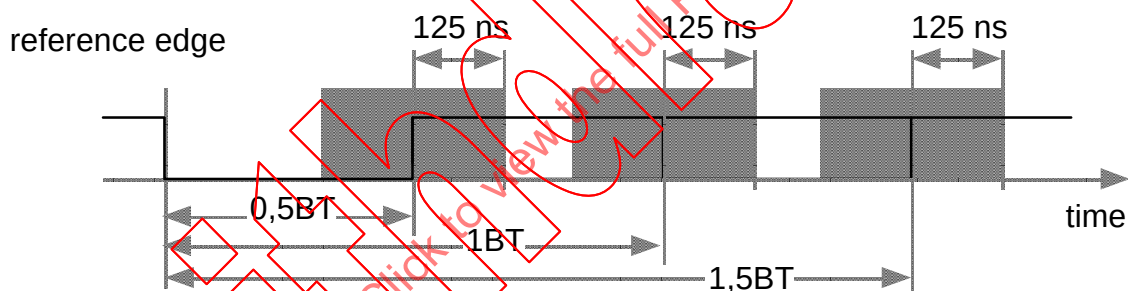


Figure 69 - Edge jitter

3.2.6.7 OGF active star coupler

Bussing of optical fibres is preferably realised by an active star coupler, as illustrated in Figure 70.

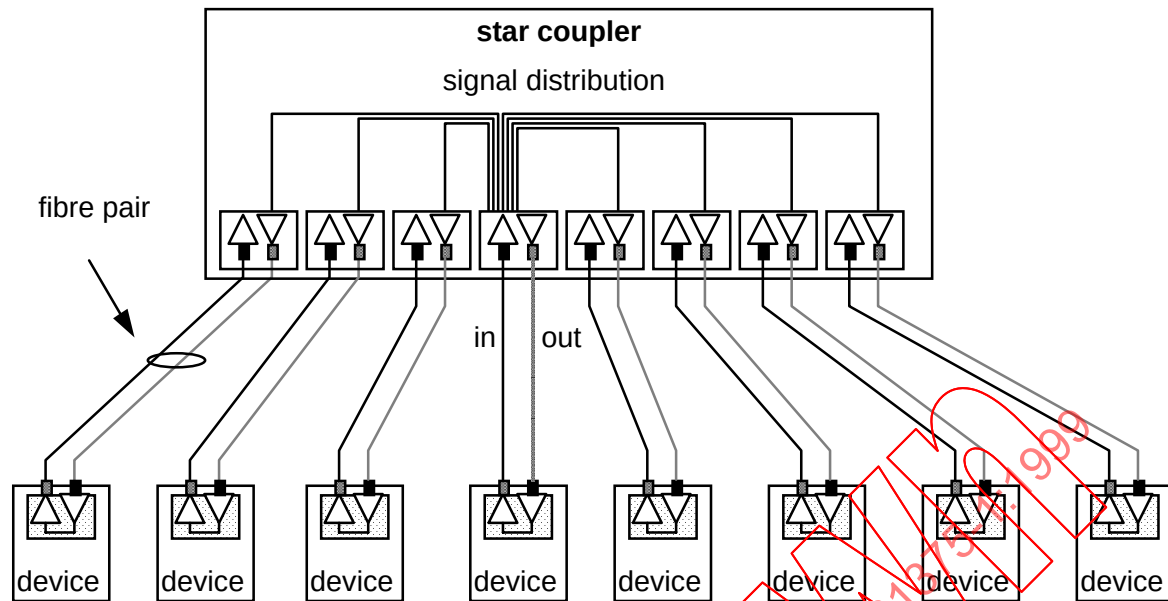


Figure 70 - Example of active star coupler

NOTE When the number of attached devices is small, bussing fibres by means of passive stars or taps is possible, provided the receiver's dynamic range permits it.

An active star coupler shall redistribute an incoming light signal over all outputs, except the output corresponding to the incoming signal.

The input and outputs of the active star shall conform to the light levels prescribed in 3.3.2.6.5 and 3.2.6.6.

The maximum delay between a change in an input signal and the corresponding change in the output signals shall be two repeater delays ($2 \cdot T_{\text{repeat}}$).

3.2.6.8 OGF double-line layout (option)

In case of a double-line optical segment, a star coupler shall be connected to either Line_A or Line_B, but not to both.

NOTE This rules out inserting devices on a star coupler used in a double-line segment.

EXAMPLE - A redundant layout with optical fibres is shown in Figure 71.

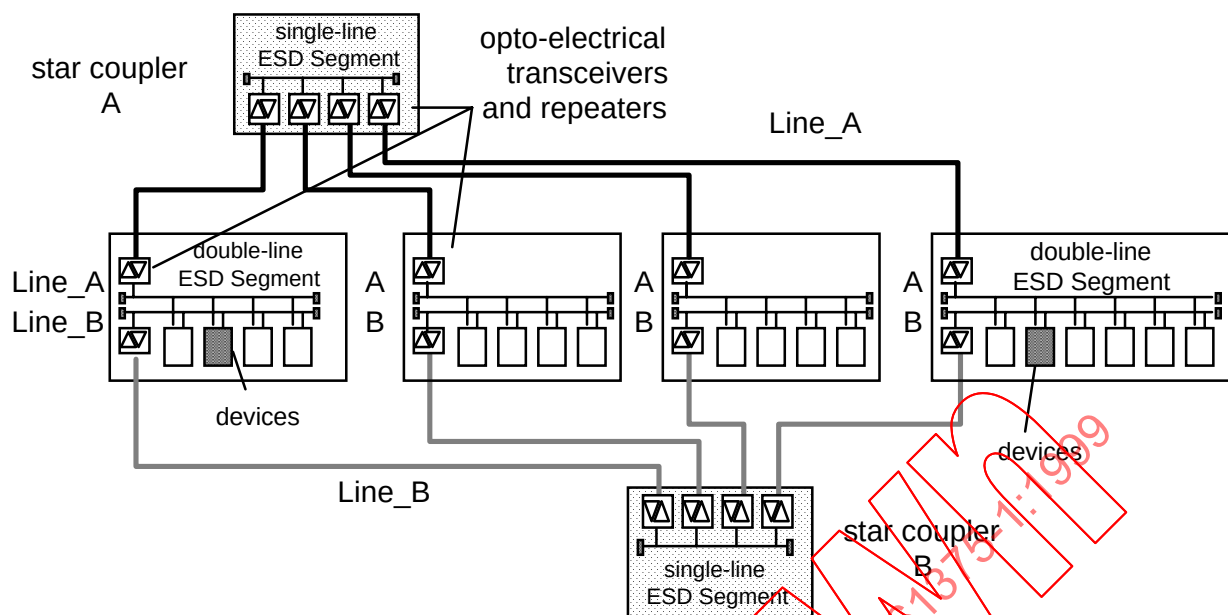


Figure 71 - Example of a duplicated star coupler

3.3 Medium-dependent signalling

This subclause specifies the signalling, which is common to electrical and optical media. Medium-dependent differences are specified.

3.3.1 Frame encoding and decoding

3.3.1.1 Conventions

Frame encoding and decoding assume a binary medium with two distinct levels, called HIGH and LOW.

The transmitter imposes these levels on the line according to its transmitter signal TxS, and the receiver detects them as the received signal RxS. The relation between the TxS signal, the line levels, and the RxS signal is defined for each medium.

The following specifications consider the TxS and RxS signals as idealised line levels.

3.3.1.2 Bit encoding

Individual data bits, "1" and "0", shall be encoded in a bit cell as follows, as shown in Figure 72:

- A "1" shall be encoded by a HIGH level during the first half of a bit cell followed by a LOW level during the second half of a bit cell;
- A "0" shall be encoded by a LOW level during the first half of a bit cell followed by a HIGH level during the second half of a bit cell.

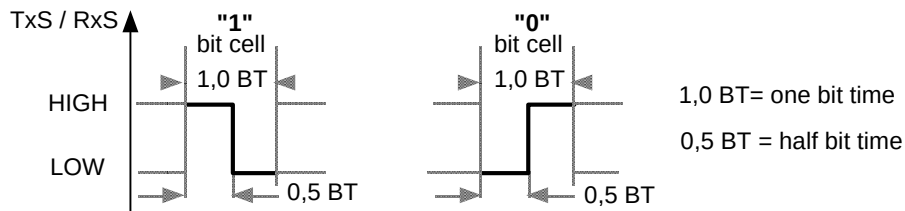


Figure 72 - "0" and "1" data encoding

NOTE This encoding is known as Manchester or Bi-phase-L encoding. There is no general agreement regarding the signal polarity.

3.3.1.3 Non-data symbols

Non_Data symbols, NH and NL, shall be encoded in a bit cell as follows, as shown in Figure 73:

- A "NH" shall be encoded by a HIGH level during one bit cell;
- A "NL" shall be encoded by a LOW level during one bit cell.

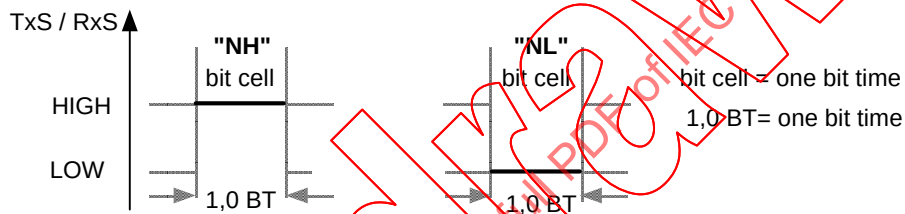


Figure 73 - Non_Data symbols encoding

3.3.1.4 Start Bit

The transmitter shall begin a frame with a Start Bit, which shall be sent as a "1" symbol.

NOTE 1 The timing of the start bit depends on the medium.

NOTE 2 The start bit may be received with a different shape or timing because of transitory effects and definition of the idle level.

3.3.1.5 Start Delimiter

Master Frames shall be headed by a Start Delimiter, called the Master Start Delimiter, consisting of the sequence:

{Start Bit, "NH", "NL", "0", "NH", "NL", "0", "0", "0"}, as shown in Figure 74.

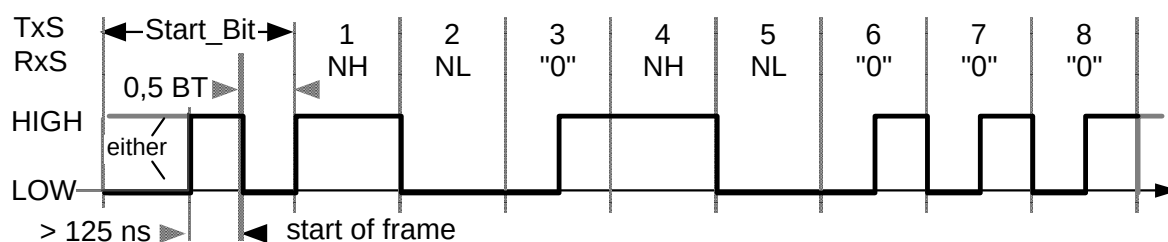


Figure 74 - Master Start Delimiter

Slave Frames shall be headed by a Start Delimiter, called the Slave Start Delimiter, consisting of the sequence:

{Start Bit, "1", "1", "1", "NL", "NH", "1", "NL", "NH"}, as shown in Figure 75.

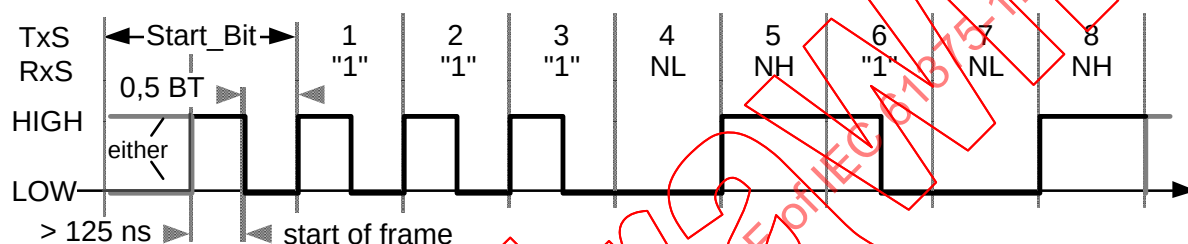


Figure 75 - Slave Start Delimiter

3.3.1.6 End Delimiter

The transmitter shall, after the last bit cell of a frame, send an End Delimiter, depending on the medium:

- with the ESD medium, append an "NL" symbol and cease transmission as specified in 3.2.4.8.4;
- with the EMD medium, append an "NL" symbol followed by an "NH" symbol and cease transmission as specified in 3.2.5.9.4;
- with the OGF medium, append an "NL" symbol and cease transmission.

NOTE The End Delimiter specification depends on the medium.

EXAMPLE - An End Delimiter for the EMD medium is shown in Figure 76.

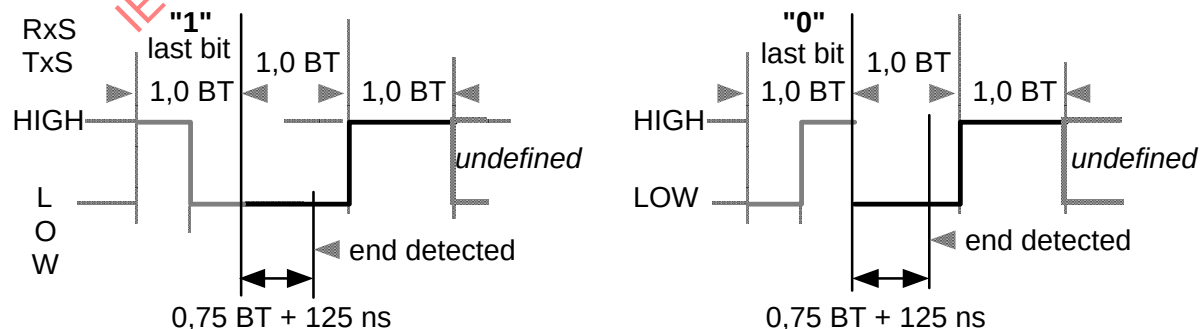


Figure 76 - Example of End Delimiter for EMD medium

The decoder shall consider that an End Delimiter is present when the RxS signal remains at the LOW level for longer than $0,75 \text{ BT} + 0,125 \mu\text{s}$.

A decoder shall ignore any pulses after the end of frame until it detects the next Start Delimiter.

3.3.1.7 Valid frame (definition)

A decoder shall recognise a valid frame as consisting of:

- a Start Delimiter (3.3.1.5); followed by
- Frame Data consisting of "0" and "1" symbols (3.3.1.2); and followed by
- an End Delimiter (3.3.1.6).

EXAMPLE- A typical valid frame with 16 bits of useful data is shown in Figure 77.

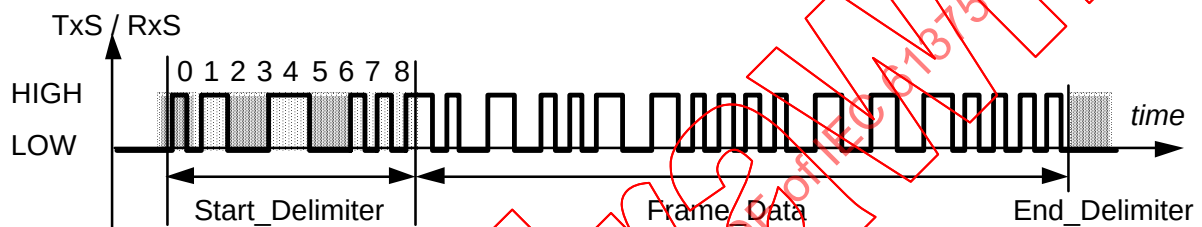


Figure 77 - Example of a valid frame (OCF medium)

The decoder may detect the correct polarity of signals by decoding the Start Delimiter, but it may not automatically invert the signal if the polarity is wrong.

To define timing:

- the start of the frame is defined as the HIGH to LOW transition of the Start Bit;
- the end of the frame is defined as the last HIGH to LOW or LOW to HIGH transition before the End Delimiter.

NOTE 1 The decoder may check in addition that the number of data bits is plausible.

NOTE 2 Since the decoder of the Trusted_Line and of the Observed_Line may check different criteria, they may interpret a frame differently.

3.3.1.8 Detection of line idle

The decoder shall have the ability to sense that the line is idle.

NOTE The decoder is expected to distinguish a limited amount of spurious noise (e.g. coming from reflections) from the absence of transmission.

3.3.1.9 Detection of collision

The decoder shall have the ability to distinguish between a line idle and a collision.

NOTE The decoder is not expected to distinguish a lasting spurious noise from a collision.

3.3.1.10 Receiver behaviour in case of error

If it detects a collision, the decoder shall ignore all frames on that line until the next Master Frame_Delimiter is received.

3.3.1.11 Jabber halt

A device may provide means to detect that its transmitter is active during a time longer than a configurable value T_{jabber} , and means to disconnect the transmitter in such a case.

In this case, the time-out T_{jabber} shall be set to the duration of the longest frame which the device is capable of sending under normal operating conditions plus 10 %.

The jabber-halt shall enable transmission again after the next

3.3.2 Line Redundancy (option)

This subclause specifies the handling of redundant lines in case this option is used.

3.3.2.1 Principle

The line redundancy principle assumes that a device transmits simultaneously the same data over both Line_A and Line_B, and that a device accepts the data from one line, called the Trusted_Line, while it monitors the other line, called the Observed_Line.

Each device chooses its Trusted_Line and Observed_Line based on the signals generated by its physical layer, or upon request of its Link Layer, independently from other devices.

The skew t_{skew} is defined as the timing difference between the signals over both lines, measured at the transmitter, at the receiver, or anywhere on the line as shown in Figure 78.

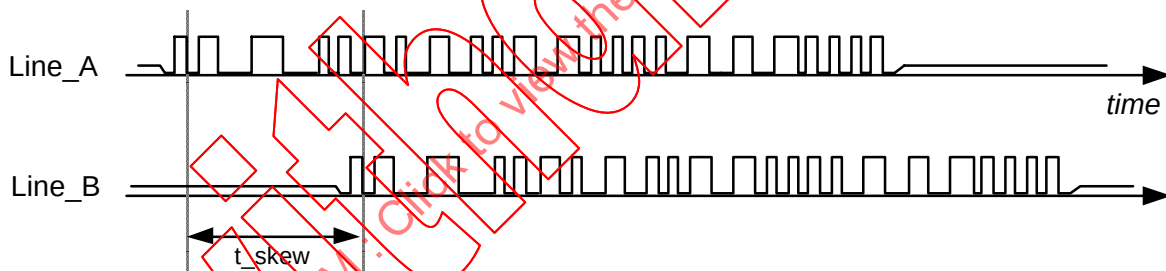


Figure 78 - Signal skew

3.3.2.2 Redundant transmission

3.3.2.2.1 Operation

A device with double-line attachment shall transmit a frame over both redundant lines Line_A and Line_B, so that the skew of the two output signals does not exceed $T_{skew\_t} = 0,100 \mu s$.

3.3.2.2.2 Propagation skew

The layout of the bus shall ensure that the skew between the signals received over Line_A or Line_B does not exceed $T_{skew} = 8,0 \mu s$ at any place on the bus.

3.3.2.3 Redundant reception

A device shall receive data from only from the Trusted_Line and shall monitor continuously the Observed_Line.

A receiver shall operate correctly and report no error if it receives valid frames on both lines with a skew less than $T_{skew\_r} = 8,0 \mu s$.

3.3.2.4 Switchover

A device shall exchange the Trusted_Line and the Observed_Line under the following conditions:

- a) if no valid frame has been received from the Trusted_Line for $T_{switchover} = 1,4 \text{ ms}$ since the end of the last valid Master Frame or since the last switchover;
- b) if the decoder of the Observed_Line detects a valid frame and the decoder of the Trusted_Line detects no valid frame within T_{skew_r} , but no collision is expected on that frame (see 3.4.2.6.2), and after the Observed_Line has been idle for longer than 2,0 BT, and provided Redundant_Line_Disturbed is not set (3.3.2.5);
- c) if a valid frame has been received over the Trusted_Line, but the frame length or the Check Sequence is incorrect, provided Redundant_Line_Disturbed is not set (option).
- d) if a device responds to a Device_Status_Request with a Device_Status_Response, provided Redundant_Line_Disturbed is not set, this condition being mandatory for Class 1 devices and a configuration option for the other Classes;
- e) if a device receives a request of its Link Layer, this condition being a configuration option not applicable to Class 1 devices.

NOTE 1 The time-out $T_{switchover}$ (1,4 ms) is related to the maximum distance between master frames T_{alive} (1,3 ms).

NOTE 3 Once a switchover took place, a switchover to the former line may only take place once the Redundant_Line_Disturbed bit has been reset by a Device_Status_Response or by the Link Layer of that device.

NOTE 4 If a continuous jammer sends on the Trusted_Line, switchover may have to wait 1,4 ms until forced switchover.

3.3.2.5 Redundancy status report

The MAU shall report the redundancy status to the Link Supervision Interface through two status bits, called LAT and RLD:

- LAT (Line_A Trusted) shall be set while Line_A is the Trusted_Line;

- RLD (Redundant_Line_Disturbed) shall be:

- set if a switchover took place or if the decoder of the Trusted_Line detects a valid frame and the decoder of the Observed_Line detects no valid frame within a skew less than T_{skew_r} , but no collision is expected on that frame (see 3.4.2.6.2);
- reset either after sending the Device_Status_Response or by the Link Layer (at the application's choice).

In a single-line attachment, a device shall consider the unused line as permanently disturbed, RLD is always set.

There shall be an error counter for each line, which shall be incremented by a Line_Disturbed signal defined as follows:

The Line_Disturbed signal shall be

- set:

- if a collision happens on the Trusted_Line, but no collision is expected (see 3.4.2.6.2);
- if a valid frame is received on the Observed_Line and no valid frame appears on the Trusted_Line within T_{skew_r} , but no collision is expected (see 3.4.2.6.2), and

- reset

- by reception of a valid frame on the Trusted_Line.

NOTE This definition of the error counter prevents the counter from being rapidly incremented if there is a permanent disturbance on a line, e.g. a line break or if the redundant line is not used.

3.3.3 Repeater

Signal regeneration takes place in repeaters (optical or galvanically isolated coupler). The following specifications aim at allowing at least up to four repeaters in series.

3.3.3.1 Repeater between single-line segments

A repeater shall fulfil the following specifications:

- a) it shall recognise the initial direction as that of the first received signal pulse and forward the signal in the opposite direction, that direction shall not change until the signal received from the initial direction remains stable for $2,0 \mu\text{s}$;
- b) it shall accept pulse distortion at the input of less than $\pm 0,125 \mu\text{s}$;
- c) it shall regenerate physical levels transparently, without interpretation of the frame contents, the only known elements being edges at intervals of 0,5 BT, 1,0 BT and 1,5 BT;
- d) it shall produce a pulse distortion at the output of less than $10,0 \text{ ns}$;
- e) If a collision takes place, the misplaced edges shall be transmitted without re-synchronisation, but collisions shall not be propagated in the initial direction;
- f) it shall introduce a delay T_{repeat} between an edge at one input and the corresponding edge at the other output of at most $3,0 \mu\text{s}$, whilst the difference between the minimum and the maximum delay shall be less than 0,20 BT, except if g) applies;
- g) it shall delay a frame which comes too close after another so that the inter-frame spacing remains above the value specified in 3.4.2.3;
- h) it shall recognise and isolate a segment in which a continuous transmitter is active for longer than a time-out $T_{\text{jabber\_all}}$ equal to the longest frame that any device on this segment is expected to send, plus 15% (3.3.1.11).

NOTE The time-out $T_{\text{jabber\_all}}$ of a repeater is larger than T_{jabber} of a device by 5% to ensure that a device is able to shut down itself in case of failure before its failure affects the segment.

EXAMPLE - The principle circuit of a repeater between two EMD segments is shown in Figure 79.

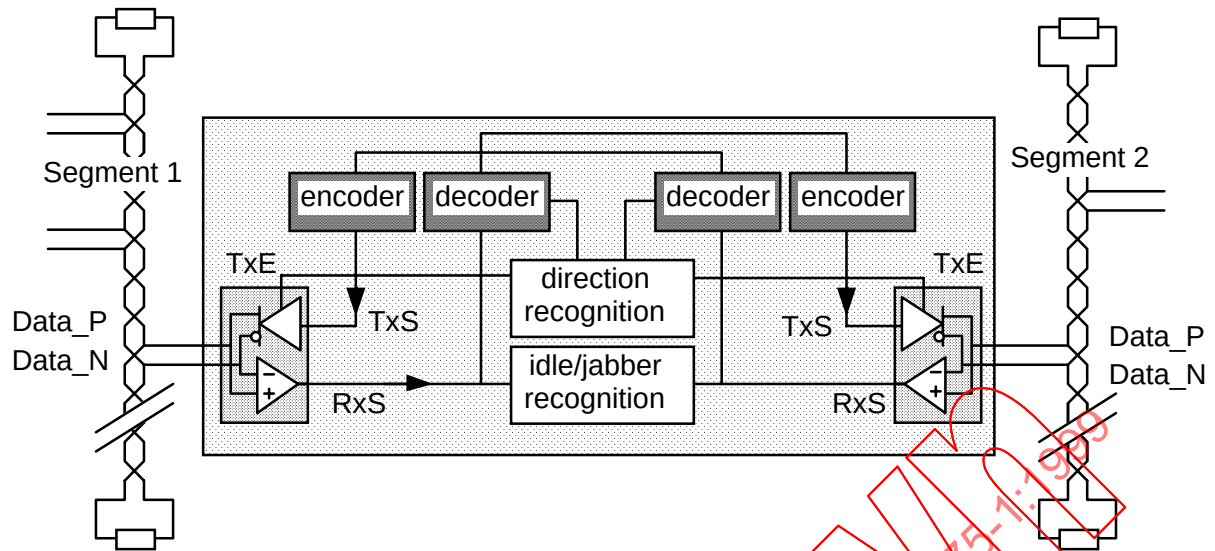


Figure 79 - Example of repeater for single-line attachment

3.3.3.2 Repeater for redundant medium (option)

If a repeater is used to connect a single-line to a double-line segment, it shall be able to implement the redundancy switchover.

EXAMPLE - The schematics of a repeater for redundant medium are shown in Figure 80.

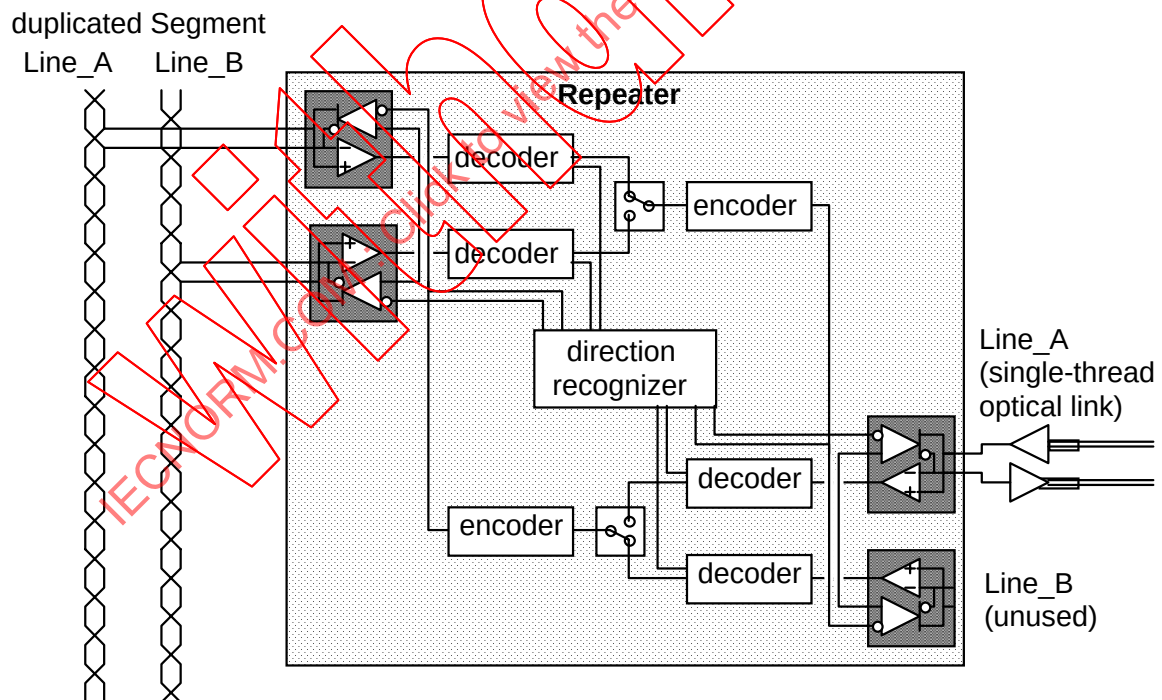


Figure 80 - Example of repeater connecting a double-line to a single line segment

3.4 Frames and Telegrams

3.4.1 Frame format

3.4.1.1 Master Frame format

A Master Frame shall, as shown in Figure 81:

- begin with the Master Start Delimiter (3.3.1.5);
- followed by 16 bits of Frame Data (3.5.2.1);
- followed by the 8-bit Check Sequence (3.4.1.3).

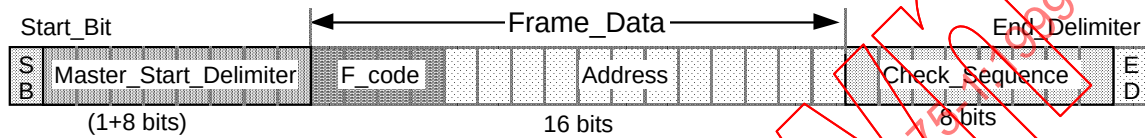


Figure 81 - Master Frame Format

3.4.1.2 Slave Frame format

A Slave Frame shall, as shown in Figure 82:

- begin with the Slave Start Delimiter (3.3.1.5);
- followed by 16, 32, 64, 128 or 256 bits of Frame Data (3.5.3.1);
- include an 8-bit Check Sequence after each word of 64 data bits or be appended to the frame when the Frame Data is 16 or 32 data bits (3.4.1.3).

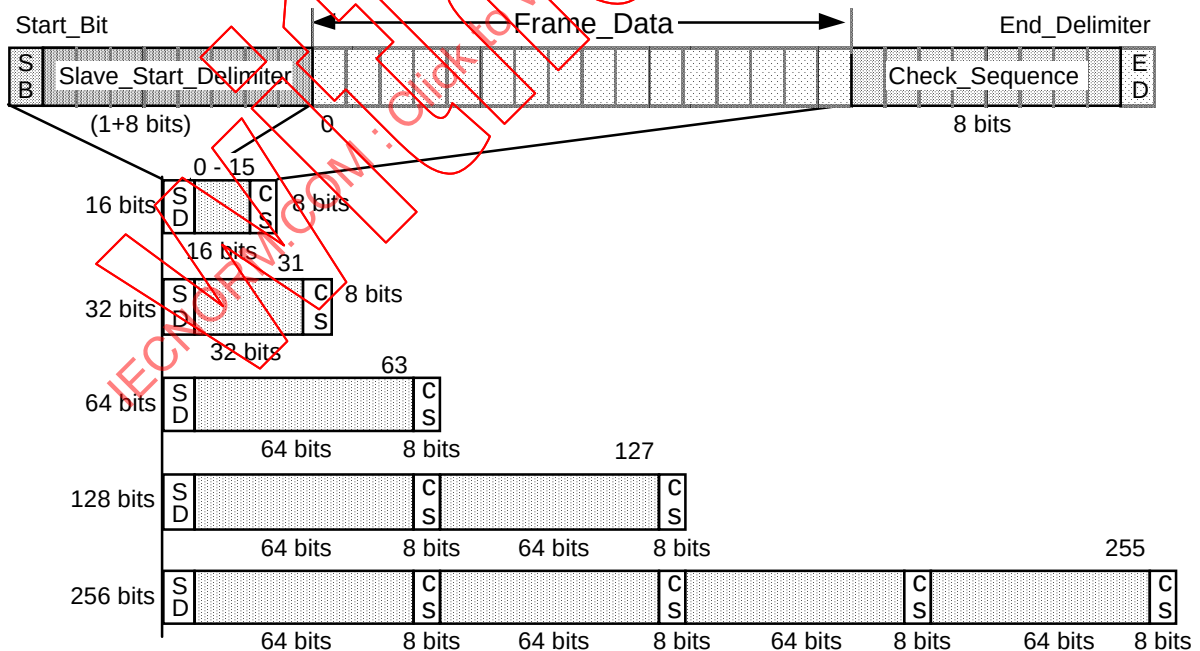


Figure 82 - Slave Frames

NOTE 1 Start Bit (SB) and End Delimiter (ED) are implicit in the following specifications and omitted in the Figures.

NOTE 2 Longer frames could be built on the same pattern, i.e. one Check Sequence after each word of 64 bits.

3.4.1.3 Check Sequence

Frames shall be protected by one or more 8-bit Check Sequences.

The contents of the data shall be treated as a codeword of 64 bits (16 and 32 bit for the smaller frames), excluding Start Delimiter and End Delimiter.

This codeword and the following Check Sequence shall be transmitted most significant bit first.

The Check Sequence shall be computed as a cyclic redundancy check (CRC) on the 16, 32 or 64 bits of data it protects.

The algorithm for the Check Sequence shall correspond to IEC 60870-5-1 format class FT2, which is repeated here.

The Check Sequence shall be computed according to the following generating polynomial:

$$G(x) = x^7 + x^6 + x^5 + x^2 + 1$$

The resulting 7-bit remainder shall be extended by an even parity bit.

All eight bits shall be transmitted inverted.

EXAMPLE:

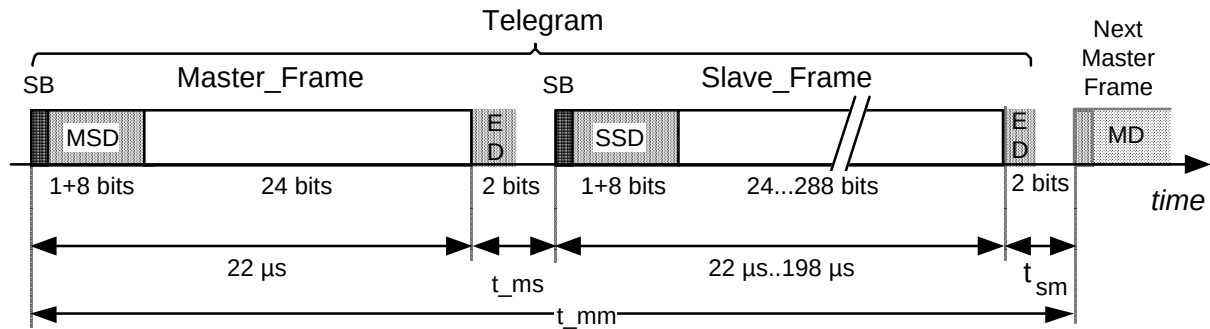
The 16-bit information	0111 1110 1100 0011
multiplied by x^7	0111 1110 1100 0011 000 0000
divided by $x^7 + x^6 + x^5 + x^2 + 1$:	1110 0101
yield a remainder of:	001 0001
which is extended by an even parity bit:	001 00010
and transmitted inverted as the check octet:	CS = 1101 1101

3.4.2 Telegram timing

3.4.2.1 Conventions

A telegram is defined as a Master Frame and of the Slave Frame sent in response to it.

EXAMPLE The timing of a telegram anywhere on the bus is shown in Figure 83.



Key

MSD Master_Start_Delimiter

SSD Slave_Start_Delimiter

SB Start_Bit

t_{ms} time from Master_Frame to Slave_Frame sent in response to it.

t_{sm} time from Slave_Frame to following Master_Frame.

t_{mm} time from Master_Frame to Master_Frame.

Figure 83 - Telegram timing

The space between frames is counted from the start of frame and end of frame points defined for each medium.

In a double-line segment, the interframe spacing is counted from the end of the first frame on any line, whichever comes later, to the start of the second frame, whichever comes earlier and as long as the RLD bit is not set.

3.4.2.2 Reply delay (definition)

The reply delay T_{reply} is, for a given bus, the maximum delay which may appear between the end of a Master Frame and the beginning of the Slave Frame sent in response to it, measured at the master.

The reply delay is the sum of propagation, decoding and access delays.

T_{reply} is a configuration parameter which tells the master how long it shall wait before sending the next Master Frame if it receives no Slave Frame or if it expects a collision (see 3.4.2.6.2).

EXAMPLE - The reply delay in a bus with 2 repeaters is shown in Figure 84 .

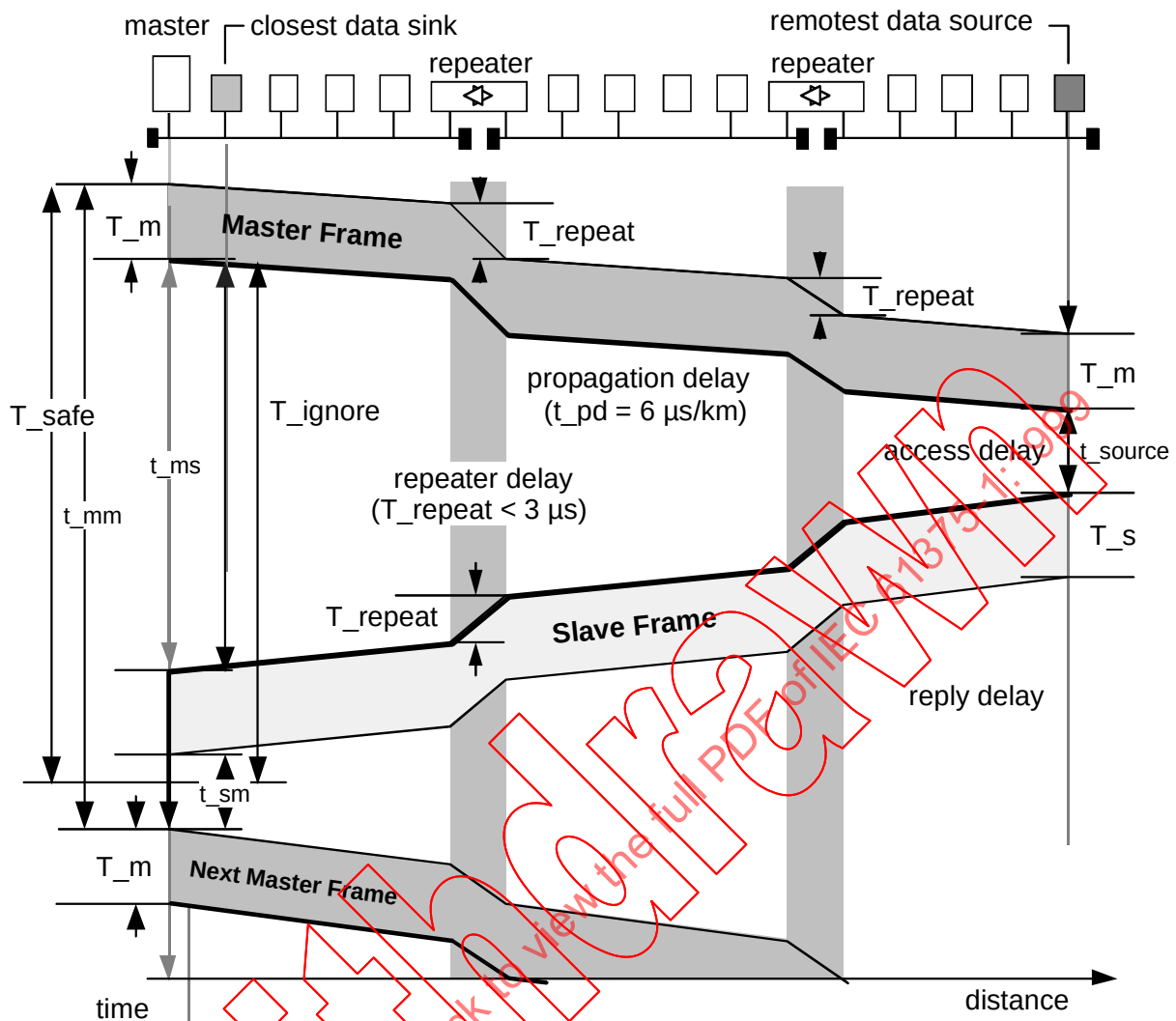


Figure 84 - Example of Reply delay

T_{reply} for a given bus is calculated as the sum of twice the propagation delays (line and repeaters), plus the delay at the slave in the worst-case:

$$T_{reply} [\mu s] = 2 \times (6,0 \times L + T_{repeat\_max} \times N_{rep}) + T_{source\_max}$$

where:

- 6,0 μs/km is the worst-case delay per km of a line (0,5 x speed of light);
- L is the electrical length of the line in kilometres (which is larger than the physical distance);
- T_{repeat_max} is the maximum delay that a repeater may introduce (3,0 μs in one direction);
- N_{rep} is the number of repeaters;
- T_{source_max} (6,0 μs) is the maximum delay at the source (3.4.2.3).

EXAMPLES:

$T_{reply} = 6,24 \mu s$ for a 20,0 m bus without repeater in between.

$T_{reply} = 44,0 \mu s$ for a 2,0 km bus with two repeaters.

The default and minimum value of T_{reply} shall be $T_{reply\_def} = 42,7 \mu s$.

NOTE In the "Extended Reply Delay" configuration option, T_{reply} exceeds the value T_{reply_def} .

3.4.2.3 Frame spacing at the source

After the end of a Master Frame, the addressed slave source device shall reply with a Slave Frame as shown in Figure 85:

- a) at the earliest after $T_{source\_min} = 2,0 \mu s$; and
- b) at the latest before $T_{source\_max} = 6,0 \mu s$.

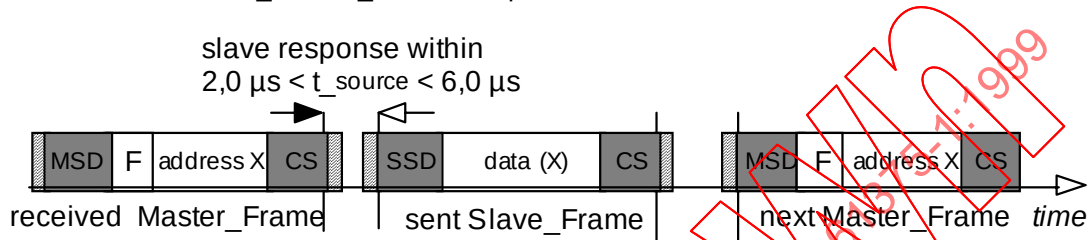


Figure 85 - Frame spacing at the source side

NOTE The recommended value of T_{source} is $4,0 \mu s$. Devices with $T_{source} > 3,0 \mu s$ always comply with the 3.2.3.2.

3.4.2.4 Frame spacing at the destination

3.4.2.4.1 Default frame spacing

A destination device shall ignore all Slave Frames which start later than T_{ignore} after the end of the last Master Frame it received, as shown in Figure 86.

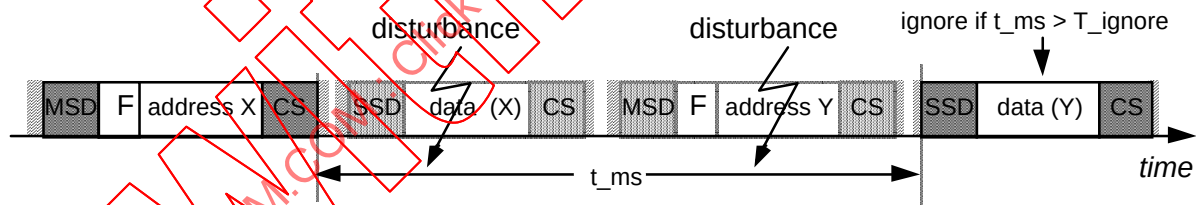


Figure 86 - Frame spacing at the destination(s)

The time-out T_{ignore} shall have a default value of $T_{reply\_def} = 42,7 \mu s$.

NOTE Process Data Slave Frames do not contain an address, but are identified by the preceding Master Frame. The loss of a Slave Frame (data(X)) and of the following Master Frame (address Y) could lead a device to accept the second Slave Frame (data(Y)) as belonging to the first Master Frame (address X). Therefore, a device suspects a double loss if it measures a delay t_{ms} longer than the time to transmit the smallest Slave Frame and the following Master Frame, which is $49 \mu s$ ($22 \mu s + 2 \mu s + 22 \mu s + 3 \mu s$). T_{ignore} is therefore set by default to a value $42,7 \mu s$, which is smaller than $49 \mu s$. On the other hand, a device should be able to accept a Slave Frame which is delayed by T_{reply} , to allow reception of worst-case delayed Slave Frames. When T_{reply} is smaller than $42,7 \mu s$ this condition is always fulfilled, but it is not for Extended Reply Delay mode, which requires special precautions (3.4.2.4.2).

3.4.2.4.2 Extended Reply Delay frame spacing

In long busses where the value of T_{reply} exceed the default value of T_{reply_def} , T_{ignore} may be increased to a value larger than T_{reply_def} , provided the following precautions are taken:

- Devices with a time-out T_{ignore} set to a value higher than $T_{\text{reply\_def}}$ shall indicate this by setting the ERD (Extended Reply Delay) bit in their Device_Status.
- The configuration variable T_{safe} shall be set equal to the highest value of T_{ignore} used by any device attached to the bus.
- The configuration variable T_{safe} shall be set equal to $T_{\text{reply\_def}}$ if all devices in a given bus are using the value $T_{\text{ignore}} = T_{\text{reply\_def}}$.

NOTE In Extended Reply Delay applications, the master is responsible to space the Master Frames so that the minimum time to transmit a Slave Frame and the following Master Frame is greater than T_{safe} .

In effect, if any device uses a value of T_{ignore} which is larger than T_{safe} , there exist a potential integrity breach on that device. To detect this situation, the bus administrator should check if any device is operating in Extended Reply Delay mode by reading its Device_Status.

3.4.2.5 Frame spacing at the master

A master shall send the next Master Frame not later than $T_{\text{alive}} = 1,3 \text{ ms}$ after the start of the previous Master Frame, as shown in Figure 87.

A master may send the next master frame:

- at the earliest T_{safe} after the start of its previous master frame in all cases (3.4.2.4.2);
- at the earliest a time $T_{\text{reply}} + T_{\text{m}}$ after the start of its previous master frame if it expects a collision or a silence in response to its sent master frame.
- at the earliest $T_{\text{sm\_min}} = 3,0 \mu\text{s}$ if it expects neither collision nor silence in response to its previous master frame.

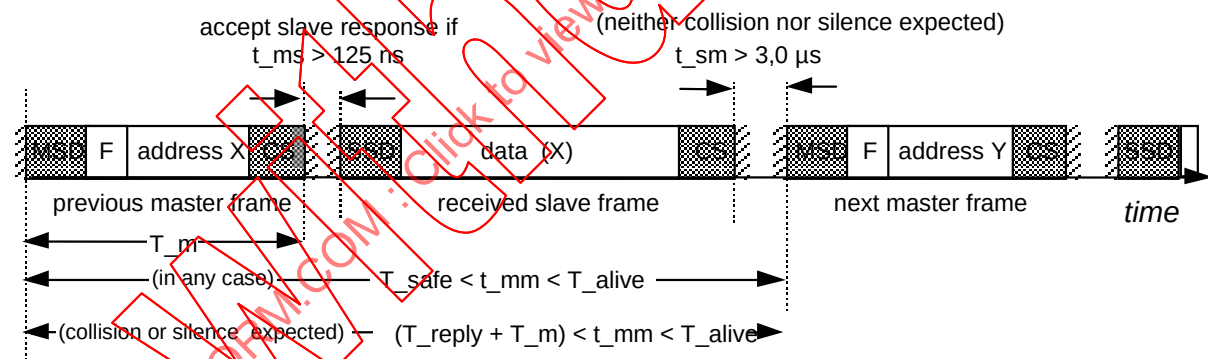


Figure 87 - Frame spacing at the master side

NOTE 1 The T_{alive} delay is used to supervise the master's activity for redundant masters.

NOTE 2 The requirement $t_{\text{mm}} > T_{\text{safe}}$ is always met when all devices on the bus use the default value $T_{\text{ignore}} = 42,7 \mu\text{s}$, since the minimum time in which a telegram can be transmitted is equal to $49 \mu\text{s}$ ($= 22,0 \mu\text{s} + 2,0 \mu\text{s} + 22,0 \mu\text{s} + 3,0 \mu\text{s}$), which is longer than $42,7 \mu\text{s}$. If any device operates in Extended Reply Delay mode, T_{safe} will be set to the highest value of T_{ignore} .

3.4.2.6 Detection of correct frames, collision and silence by the master

3.4.2.6.1 Correct frame (definition)

A correct Slave Frame is defined as a valid frame (3.3.1.7) which is received within the limits set by 3.4.2.4, headed by a Slave Start Delimiter, which has a size corresponding to that announced in the Master Frame and with correct CRCs inserted.

NOTE Only correct frames may be forwarded to the application.

3.4.2.6.2 Detection of collision by the master

The master shall detect a collision if the decoder of the trusted line reports a collision or if the received slave frame is not correct.

The master shall detect no collision if it receives a correct slave frame, even if a second valid frame comes within T_{reply} after the first one due to long propagation delays. The master shall then consider the first received correct Slave Frame as a single answer.

A collision is said expected on the frames of type General_Event_Response, Group_Event_Response and Individual_Event_Response.

NOTE 1 A collision occurs when several transmitters are simultaneously active. This occurs normally only during event arbitration. The collision is monitored only by the master.

NOTE 2 In a long bus, it is possible that the master receives a correct frame although several slaves answer: a close transmitter can override a remote one, or a second frame can arrive after the end of the first frame because the reply delay T_{reply} could exceed the minimum Slave Frame duration ($= 22,0 \mu s$).

3.4.2.6.3 Detection of silence by the master

The master shall detect a silence if the decoder reports neither a valid frame nor a collision within T_{reply} after the end of the previous Master Frame.

A silence is said expected on all frames except Process Data_Reponses.

NOTE A disturbance can cause the master to interpret silence as a collision. This can lead the event search on the false path. The decoder helps to distinguish between disturbance and collision.

3.5 Link Layer Control

3.5.1 Addressing

3.5.1.1 Device Address

A device shall be identified on a bus by its Device Address.

The Device Address shall have a size of 12 bits.

Device Address 0 shall not be used.

NOTE 1 Devices supporting the Message Data capability should have a Device Address smaller than 256.

NOTE2 The broadcast device address is indicated by the mode field, not by a dedicated address.

3.5.1.2 Logical_Address

A Process Data source shall be identified by its Logical_Address.

The Logical_Address shall have a size of 12 bits.

Logical_Address 0 shall not be used.

3.5.1.3 Group_Address

For the purpose of Event_Arbitration, devices may be addressed by their Group_Address, as specified in 3.6.3.1

3.5.2 Master Frames contents

3.5.2.1 Master Frame format

The Master Frame shall carry a 16-bit data word.

The first transmitted bit of that word shall be its most significant bit, identified as bit 0 as shown in Figure 88.

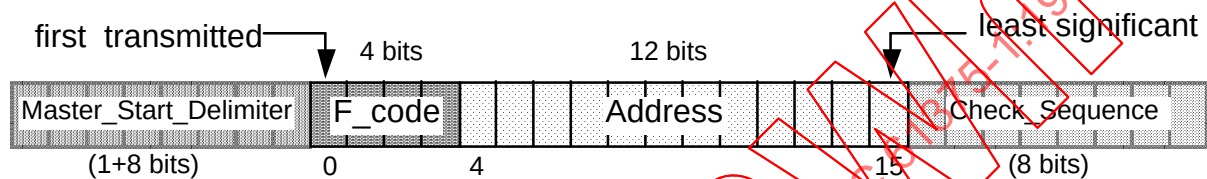


Figure 88 - Master Frame contents

The four most significant bits of the 16-bit word shall be the F_code.

The least significant 12 bits shall represent an address or parameters as specified by the F_code.

3.5.2.2 F_code encoding

The 4-bit F_code qualifies the following 12-bit and indicates the size of the expected Slave Frame.

The F_code shall be encoded according to Table 53.

Table 53 - Master Frame types and F_code

Master Frame			Slave Frame			
F_code	address	Request	source	size (bits)	response	destination
0	logical	Process Data	single device subscribed as source	16	Process Data (application-dependent)	all devices subscribed as sink
1				32		
2				64		
3				128		
4				256		
5		Reserved				
6		Reserved				
7		Reserved				
8	device	Mastership_Transfer	proposed master	16	Mastership_Transfer	master
9	all devices	General_Event (parameters)	one or more devices	16	Event_Identifier	master
10	device	Reserved		-		
11	device	Reserved	-	-		
12	device	Message Data	single device	256	Message Data	selected device(s)
13	device group	Group_Event	one or more devices	16	Event_Identifier	master
14	device	Single_Event	single device	16	Event_Identifier	master
15	device	Device_Status	single device	16	Device_Status	master or monitor device

3.5.3 Slave Frame contents

3.5.3.1 Slave Frame format

A Slave Frame shall transmit 1, 2, 4, 8 or 16 words of 16 bits each.

Bits within a word shall be transmitted most significant bit first (treating the word as an unsigned, 16 bit integer, the first transmitted bit having offset 0).

Words shall be numbered starting with 0, as shown in Figure 89.

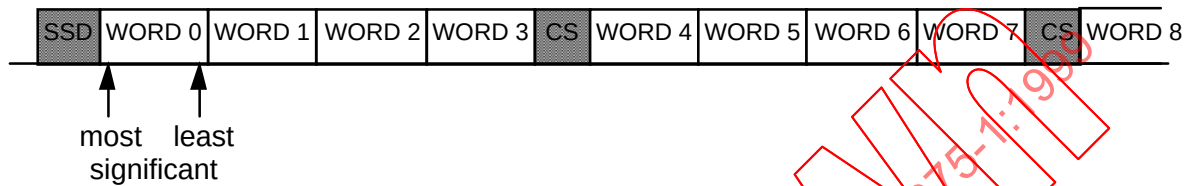


Figure 89 - Word ordering in a Slave Frame

Frames shall be extended by "0s" if the full frame size is not used for link data.

NOTE The contents of a Slave Frame are not specified except for the Message Data header and for the Supervisory Data frames.

3.5.3.2 Size error

A destination shall ignore a Slave Frame if its length does not match the length indicated by the F_code in the Master Frame, even if its Check Sequence is otherwise correct.

3.5.4 Telegram types

3.5.4.1 Process Data telegram

3.5.4.1.1 Process Data format

A master shall request the transmission of Process Data by sending a Process Data_Request, which is a Master Frame with one of the F_codes: 0..4.

NOTE F_codes 5..7 are reserved for future extensions of Process Data.

The slave which sources the Process Data corresponding to the identifier in the Process Data_Request shall respond with a Process Data_Response, as shown in Figure 90.

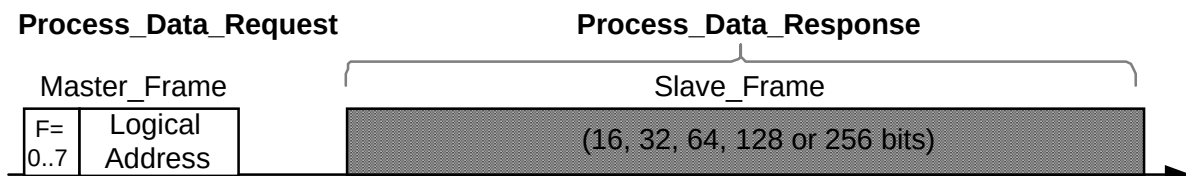


Figure 90 - Process Data telegram

3.5.4.1.2 Process Data identifier

The Process Data shall be identified by both the 12-bit Logical_Address and the F_code in the Master Frame, which indicates the size of the Process Data (3.5.2.2).

For each Logical_Address, the size of the corresponding Process Data shall be configured in the source and all sink devices before regular operation begins.

3.5.4.1.3 Buffers

A device shall hold a buffer for each Logical_Address it is subscribed to, as source or as sink.

3.5.4.1.4 Source devices

A device sourcing Process Data, the identifier of which appears in the Process Data_Request, shall read the contents of its source buffer and send it in one indivisible operation in the Slave Frame.

A device shall not respond if the F_code of the Process Data_Request indicates a frame length different from the configured length for that Logical_Address.

A source buffer shall provide means to be overwritten with "0" to invalidate its data.

3.5.4.1.5 Sink devices

A device sinking Process Data, the identifier of which appears in the Process Data_Request, shall store that frame in the corresponding buffer, overwriting the previous contents of that buffer in one indivisible operation with the contents of the frame received over the bus.

A device sinking a Process Data frame, the identifier of which appears in the Process Data_Request, shall not accept a Slave Frame if the F_code indicates a frame length different from the configured length for that Logical_Address.

A sink buffer shall provide means to indicate how long ago it was updated.

3.5.4.2 Message Data

3.5.4.2.1 Message Data format

The master shall request the transmission of Message Data by sending a Message Data_Request, which is a Master Frame with F_code = 12.

The device addressed in the Master Frame shall respond with a Message Data_Response, with the following format, starting with the first transmitted bit, as shown in Figure 91:

- a) The first 4 bits define the communication mode:
 - Mode '0001'B (+1) specifies the single-cast addressing mode,
 - Mode '1111'B (-1) specifies the broadcast addressing mode,
 - all other modes are reserved for future use;
- b) The next 12 bits define the address of the Destination_Device.

In case a broadcast mode is used, this field may be used by the application;

- c) The next 4 bits define the Protocol_Type. The Protocol_Type for the Real-Time Protocols is defined in 3.8.3.4;
- d) The next 12 bits indicate the address of the Source_Device;
- e) The next 8 bits (SZ) express the number of significant octets in the following Link Data;
- f) The following bits carry the Link Data;
- g) The remaining octets of the frame are filled with "0" if the Link Data are less than 216 bits in size.

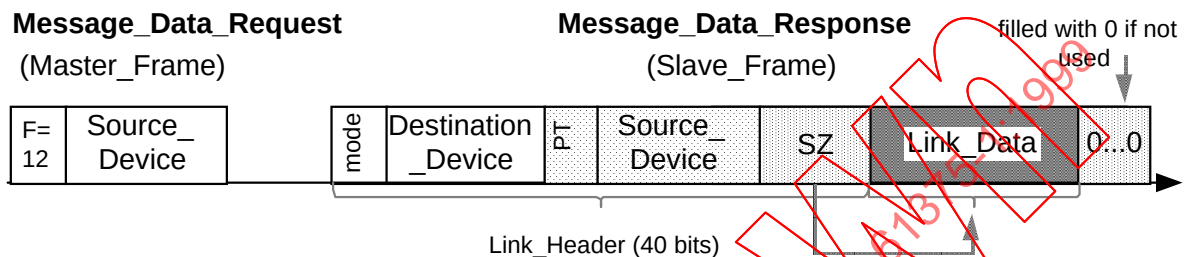


Figure 91 - Message Data telegram

NOTE 1 The Link Data contents are defined in Clause 2 of this Standard "Real-Time Protocols".

NOTE 2 The single-cast Mode and the broadcast Mode are the 2's complement of each other. For instance, if a future Mode expresses single-cast by 3 ('0011'B), the corresponding broadcast Mode will be -3 ('1110'B).

3.5.4.2.2 Queues

Each device capable of Message Data communication shall hold a pair of queues, one for sending (Send_Queue) and one for receiving (Receive_Queue) Message Data.

NOTE The queues are not prioritised.

3.5.4.2.3 Message Data protocol

When receiving a Message Data_Request, a device specified by the source address in the Message Data_Request shall:

- send a Message Data_Response from its Send_Queue and remove it from the Send_Queue or
- not respond at all if its Send_Queue is void;

When receiving a Message Data_Response, a device which recognises its own Device Address as Destination_Address or the broadcast mode shall:

- append the Message Data to its Receive_Queue, if this is possible, or
- ignore the Message Data, if this is not possible.

NOTE Repetition of lost frames is responsibility of the Transport_Layer.

3.5.4.3 Supervisory Data telegrams

3.5.4.3.1 Supervisory Data format

A master shall request Supervisory Data from a device or shall send Supervisory Data to a device with a Supervisory_Data_Request, which is a Master Frame using one of the following F_codes (the appended 12 bits are specified in the corresponding Subclause):

F_code =	8	Mastership_Transfer_Request	(3.7.2.3.2);
F_code =	9	General_Event_Request	(3.6.3.4.1);
F_code =	13	Group_Event_Request	(3.6.3.4.2);
F_code =	14	Single_Event_Request	(3.6.3.4.3);
F_code =	15	Device_Status_Request	(3.6.4.1.1).

The Supervisory_Data_Response shall carry 16 bits of Frame Data, as shown in Figure 92. These bits are specified in the Subclause defining the response frames corresponding to the F_code:

F_code =	8	Mastership_Transfer_Response	(3.7.2.3.2);
F_code =	9	Event_Identifier_Response	(3.6.3.4.4);
F_code =	13	Event_Identifier_Response	(3.6.3.4.4);
F_code =	14	Event_Identifier_Response	(3.6.3.4.4);
F_code =	15	Device_Status_Response	(3.6.4.1.2).

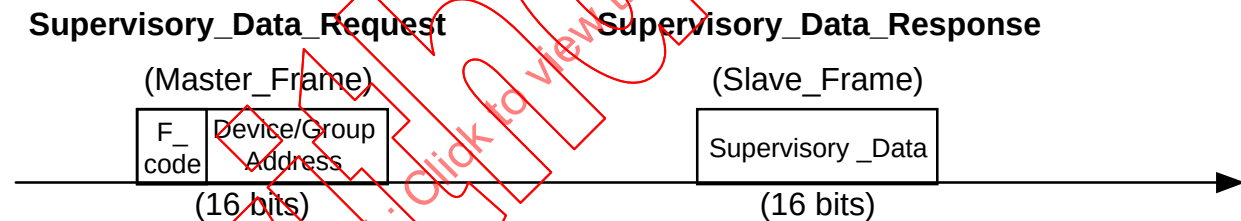


Figure 92 - Supervisory Data telegram

3.5.4.3.2 Supervisory Data protocol

Supervisory Data telegrams shall not be repeated immediately if no Supervisory_Data_Response comes after a Supervisory_Data_Request.

3.6 Medium allocation

3.6.1 Organisation

3.6.1.1 Turn

The medium access shall be controlled by one of the bus administrators, the master, which remains in control of the bus for the duration of a turn.

At the end of a turn, the master shall try to pass mastership to another bus administrator or shall obtain it again if it finds no available bus administrator.

NOTE The beginning of a turn may be observed by the attached devices, which can monitor which bus administrator is currently master, for synchronisation purpose.

3.6.1.2 Basic Period

The master shall divide its turn into fixed time slots, called "Basic Periods", all of the same duration T_{bp} .

The Basic Period to be used for a turn shall take a value:

$$1,0 \text{ ms} \leq T_{bp} \leq 2,5 \text{ ms}.$$

NOTE 1 For synchronism purposes, any fraction of the period of the controlled process, given dividing this period by a power of two and included in the range between 1 and 2 ms can be used, for example: 1,042 ms ($16 \times 60 \text{ Hz}$), 1,250 ms ($16 \times 50 \text{ Hz}$) and 1,875 ms ($32 \times 16,66 \text{ Hz}$).

A Basic Period shall be divided into four phases, as shown in Figure 93:

- a) a Periodic Phase;
- b) a Supervisory Phase;
- c) an Event Phase;
- d) a Guard Phase.

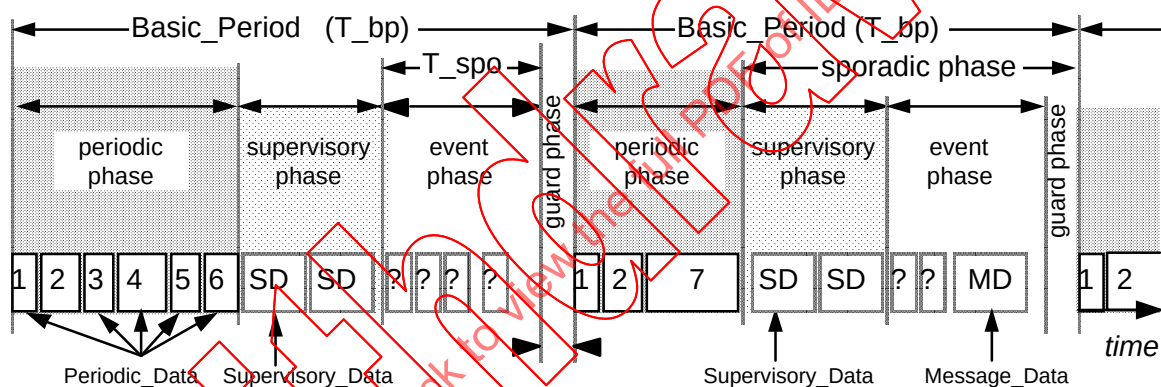


Figure 93 - Basic Periods

NOTE 2 Event Phase, Supervisory Phase and Guard Phase form the Sporadic Phase.

The value of T_{spo} may vary from Basic Period to Basic Period.

The default value of T_{spo} shall be 350,0 μs .

If it becomes necessary to use a lower value of T_{spo} for a limited number of Basic Periods, T_{spo} shall be larger than the time to transmit the shortest telegram, $T_{mm\_min} = 49,0 \mu\text{s} + T_{reply}$.

NOTE 3 The maximum value of T_{spo} is equal to the duration of the Basic Period.

3.6.1.2.1 Periodic Phase

During the Periodic Phase, the master shall perform the Periodic Polling specified in 3.6.2

3.6.1.2.2 Event Phase

During the Event Phase, the master shall perform the Event Polling specified in 3.6.3.

3.6.1.2.3 Supervisory Phase

The master shall use the Supervisory Phase for the Devices_Scan (3.6.4) and for Mastership Transfer (3.7).

NOTE Supervisory Data may be transmitted as Periodic_Data, as Sporadic_Data or by initiative of the master.

3.6.1.2.4 Guard Phase

The master shall reserve a Guard Phase as a buffer after the sporadic transmissions to allow an exact start of the next Periodic Phase.

The master shall not send a Master Frame if the transmission of the longest expected response would take longer than the time remaining until the next Basic Period is scheduled.

3.6.1.3 Padding

Each Basic Period shall be constructed in such a way that the master has an opportunity to issue a Master Frame (of any type) at least once every T_{alive} (3.4.2.5) = 1,3 ms to allow each device to monitor the line (3.7.2.2).

3.6.2 Periodic Polling

3.6.2.1 Periodic List

The Periodic List shall define the Master Frames sent out periodically and the time left for the Sporadic Phase for each Basic Period of a turn.

During Periodic Polling, the master shall send a predefined sequence of Master Frames according to its Periodic List.

Modifications made to the Periodic List shall not come into effect during the same turn.

NOTE The Periodic List contains the Periodic_Data, i.e. mainly Process Data_Requests, but also Supervisory_Data_Requests or Message Data_Requests for special purposes.

3.6.2.2 Individual Period

The master shall poll each Periodic_Data with its Individual Period T_{ip} .

An Individual Period shall be equal to the Basic Period multiplied by a power of 2, but shall not exceed 1024 T_{bp} , or in general:

$$T_{\text{ip}}(i) = T_{\text{bp}} \times 2^n, \text{ with } n = (0..10).$$

The longest Individual Period in the Periodic List, called the Macro_Period, shall not exceed 1024,0 ms.

NOTE 1 The Individual Period is defined by the application.

NOTE 2 If the shortest Individual Period exceeds the Basic Period, the Periodic Phase of some Basic Periods could be void.

3.6.2.3 Periodic Phase construction

The Periodic List is configured by the application. The following Subclauses indicate how the Periodic List is to be constructed.

3.6.2.3.1 Conventions for constructing the Periodic List

The following conventions shall apply to the format of the Periodic List as specified in the Link Supervision Interface:

- a) Periodic_Data are ranked by their Individual Period $T_{ip(i)}$;
- b) a Cycle groups Periodic_Data with the same Individual Period and receives the same name as its Individual Period in multiple of the Basic Period (e.g. Cycle_2 when polled every 2 ms and the Basic Period is 1 ms);
- c) a Macro_Cycle consists of all cycles which fill a Macro_Period, therefore it takes:
 - 1024 Basic Periods if $T_{bp} = 1$ ms or
 - 512 Basic Periods if $T_{bp} = 2$ ms;
- d) Basic Periods are numbered BP(j), where:
 - $j = (0 \dots \text{macro}-1)$,
 - BP(0) = first period of a Macro_Cycle,
 - BP (macro-1) = last period of a Macro_Cycle;
- e) a turn is defined as the number of Macro_Cycles after which a mastership transfer shall take place. A turn may be stopped after each Macro_Cycle by a Link Supervision Interface command;
- f) a Cycle may be split over several Basic Periods, into Subcycles. Cycle_n has n Subcycles, for instance Cycle_16 has 16 Subcycles, of which some or all may be void. A Subcycle is identified by its index, for instance Cycle_4.1 being the second Subcycle of Cycle_4;
- g) the last Sporadic Phase of every Macro_Cycle is reserved for mastership transfer, even if not used for mastership transfer in this Macro_Cycle.

NOTE There are no constraints about the split of Subcycles over the different periods, but for performance reasons, it is recommended to spread the cycles evenly over the periods.

3.6.2.3.2 Construction of a Macro_Cycle

The Periodic List shall contain two sub-lists for each Macro_Cycle:

- a) the Cycle_List, containing the Master Frames belonging to each of the 11 Cycles (Cycle_1 to Cycle_1024) and
- b) the Split_List, containing the number of frames of each cycle to send in each Subcycle.

EXAMPLE - A simplified construction of a Macro_Cycle consisting of 8 Cycles is shown in Figure 94.

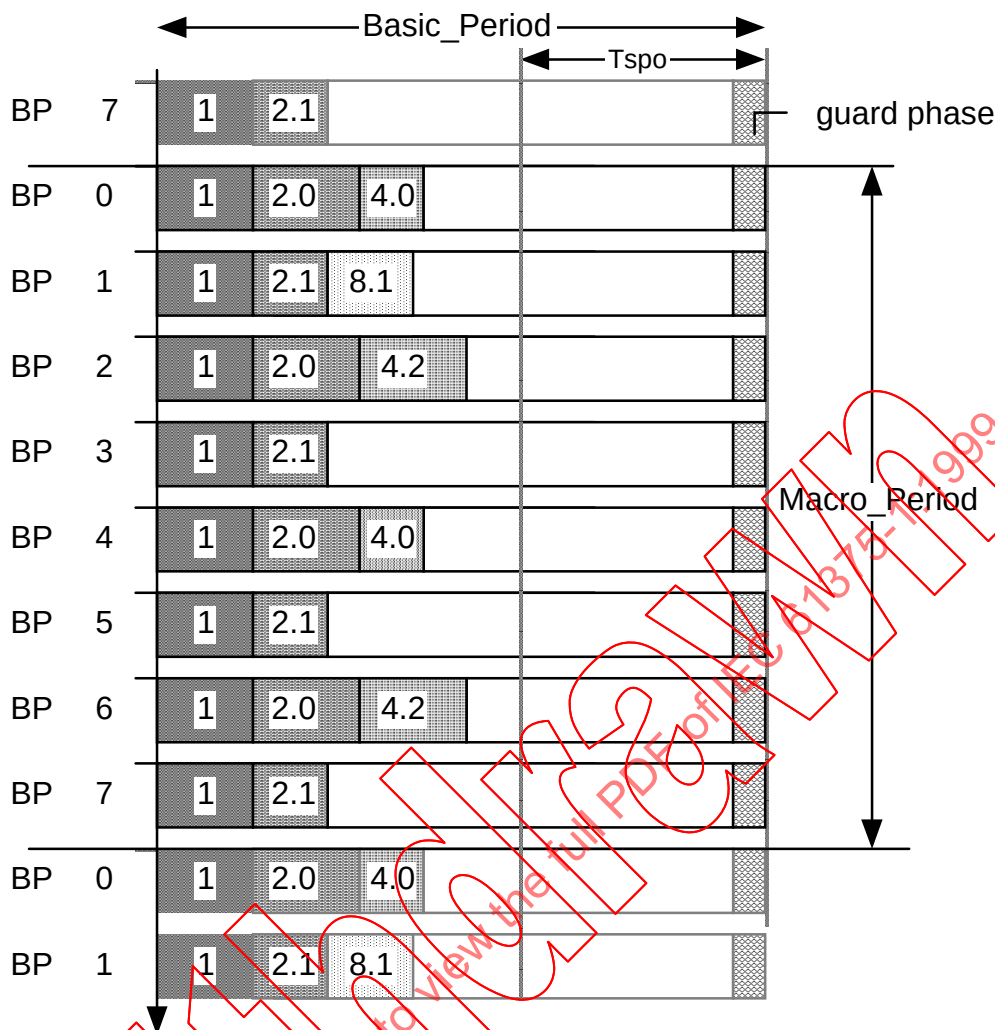


Figure 94 - Example of construction of the Macro_Cycle

The Periodic List of the bus administrator of this example contains:

- the four Cycle_Lists, one for each cycle 1, 2, 4 and 8;
- the three Split_List indicating the number of frames for:
 - Subcycles 2.0 and 2.1 (Cycle 1 has no split, it is present in every period),
 - Subcycles 4.0, 4.1 (void here), 4.2 and 4.3 (void here),
 - Subcycles 8.0, 8.1, 8.2, 8.3, 8.4, 8.5, 8.6 and 8.7 (all voids here except 8.1).

3.6.3 Event Polling

3.6.3.1 Group_Address

NOTE For the purpose of Event_Arbitration, the master selects groups of devices which share the same lowest significant address bits.

Devices on the bus which differ in the most significant bits of their Device Address form a Group, which is identified by its Group_Address.

The Group_Address is characterised by:

$G = \{M, C\};$

- M being the number of most significant digits in which the device addresses differ,
- 2M being the number of possible devices in the group ($M = 1..12$);
- C being the binary number expressed by the common least significant bits of the address.

EXAMPLE: the devices with binary address:

1111 0000 0101,
1000 0000 0101 and
0101 0000 0101

belong to the same group $\{M = 4, C = 5\}$.

3.6.3.2 Event_Round

An Event_Round collects all event requests of a given priority pending at the time the Event_Round begins.

3.6.3.2.1 Start

The master shall start an Event_Round by sending a General_Event_Request, which defines.

- a) that this is a new Event_Round and not a continuation {new_round}
- b) that an answer is required {answer_now}
- c) one of two event priority levels for this Event_Round {high_priority or low_priority}.

This General_Event_Request addresses Group $\{M = 12\}$, i.e. all possible 4096 devices.

3.6.3.2.2 Event clearing and enabling

A device shall participate in an Event_Round if it has an event of the corresponding priority pending at the time it receives a General_Event_Request {new_round}.

A participating device shall respond with its Event_Identifier_Response to an Event_Request with the following F_codes:

- a) F_code 9: General_Event_Request (if "answer_now" and priority matches);
- b) F_code 13: Group_Event_Request (if the Group_Address matches that of this device);
- c) F_code 14: Single_Event_Request (if the Device Address matches that of this device).

A device shall clear the corresponding event when it receives an Event_Read_Request which matches the F_code and address of the Event_Identifier_Response.

NOTE 1 A slave is not informed that the master has read its Event_Identifier_Response.

NOTE 2 Several devices can report the same event. In this case, all devices with that event clear it when the event is read since no one knows that it was the one which signalled that event.

3.6.3.2.3 Recognised situations

An Event_Request is defined as one of General_Event_Request, Group_Event_Request, Single_Event_Request.

The master shall distinguish the following situations occurring in response to an Event_Request, within the limits of T_reply:

- a) correct: a correct Slave Frame with 16 bits was received (3.4.2.6.1);
- b) silence: no frame has been received (3.3.1.8);
- c) collision: all other cases (3.3.1.9).

3.6.3.2.4 Event search algorithm

In case a collision comes in response to a General_Event_Request, the master shall search all pending events by sending Group_Event_Request or Single_Event_Request, with the aim of restricting the number of devices participating in the poll.

NOTE - The algorithm for searching the pending events is not specified. A general method is given in 3.6.3.3 as a recommended practice, but other algorithms are permitted. In particular, algorithms may take advantage of the fact that the identity of participating devices is known and that events tend to occur in bursts for Message Data transfers.

3.6.3.2.5 Event_Arbitration termination

The master shall terminate Event_Arbitration:

- a) when it receives an Event_Identifier_Response;
- b) when it detects either a collision or a silence in response to an Single_Event_Request.

In configurations where the event-signalling devices have a limited address range (e.g. 1..255), the master may consider that the search terminates when no response comes to a Group_Event_Request which addresses that device group.

3.6.3.2.6 Event_Round interruption and continuation

The high priority Event_Round shall be suspended only by the Periodic Phase.

In case both a high and a low priority are configured, a high priority Event_Round shall be started after each time a low priority event has been read, suspending the low priority Event_Round.

The low-priority Event_Round shall be resumed with a General_Event_Request (NOT new_round, answer_now).

3.6.3.2.7 Event_Round termination

When the search tree has been run completely, and after all events have been read, the master shall issue a General_Event_Request {NOT new_round, answer_now} to check for devices left out in the Event_Round.

If no answer comes to that General_Event_Request, the master shall close the Event_Round.

Otherwise, it shall resume the Event_Round from the beginning.

If the same event is reported more than twice during Event_Round termination, the master shall close the Event_Round and report an error.

NOTE Devices may be left out in a round due to a wrongly interpreted situation, in particular:

- undetected collision (correct frame received although several devices transmitted);
- undetected silence (disturbance caused a collision to appear when there is none);
- error during transmission of the Event_Identifier_Response;
- two or more devices requesting the same event.

3.6.3.2.8 Event_Read

The master shall re-send the received Event_Identifier_Response (Slave Frame) as an Event_Read_Request (as a Master Frame).

The master shall not repeat Event_Read_Request even if it does not receive a correct Event_Read_Response in response to it, e.g. in case of transmission error.

NOTE The master may send an Event_Read_Request just after receiving an Event_Identifier_Response, or it may bundle the Event_Read_Requests of several received Event_Identifier_Responses after the end of the Event_Round, as the application decides.

3.6.3.3 Recommended event search algorithm

The MVB Tutorial (Clause A.3) describes the event search algorithm. The following is a recommended algorithm.

3.6.3.3.1 First Event_Arbitration

If a collision occurs in response to a General_Event_Request, the master executes an Event_Arbitration to single out one event.

To this effect, the master sends a Group_Event_Request, addressing Group {N-1, 0}.

NOTE The first Group_Event_Request addresses the 2048 devices with an even address.

3.6.3.3.2 Down Event_Arbitration

Assuming that the master sent a Group_Event_Request to Group {N, C}:

- If a single response comes, the master returns that response;
- If collision occurs, the master issues a Group_Event_Request to a group half the size of the former group, and whose common address is the same:

$$G\_new = G\_old \{M = M-1, 0 + C\}$$

NOTE 1 This Group_Event_Request addresses the one half of the preceding group, whose common most significant address bit is "0".

- If silence follows a Group_Event_Request, the master issues a Group_Event_Request to a group potentially half the size of the former Group, and whose common address is:

$$G\_new = G\_old \{M = M-1, (2^{(12-M)} + C)\}$$

NOTE 2 This Group_Event_Request assumes that the event comes from the other group, and starts with the half-group whose common most significant common address bit is "0".

The master sends a Single_Event_Request where a collision occurs on the second most significant Device Address bit (Group_Event_Request with bit 4=0).

3.6.3.3.3 Up Event_Arbitration

The master continues the next Event_Arbitration at one level lower in the tree from the level where the most recent collision occurred.

To this effect, the master sends the Group_Event_Request with the least significant bit of the Group_Address set to the opposite value.

$$G\_new = G\_old \{M = M/2, 2^{(12-M)} + C\}$$

If this Group_Event_Request is responded by a silence or an Event_Identifier_Response, the master sends a General_Event_Request and restart arbitration from the top.

If the Group_Event_Request is responded by a collision, the master increases the size of the Group_Address by one and performs an Event_Arbitration on that branch:

$$G\_new = G\_old \{M = M+1, (2^{(12-M)} + C)\}$$

3.6.3.4 Supervisory Data frames for Event_Arbitration

3.6.3.4.1 General_Event_Request

The General_Event_Request shall consist of four fields of 4 bits each, as shown in Figure 95:

The bits 0 to 3 are the F_code = 9;

The next 4 bits are the Event_Mode, encoded as:

- EM = '0000'B: answer_now, NOT new_round
- EM = '0001'B: answer_now, new_round
- EM = '0010'B: NOT answer_now, NOT new_round
- EM = '0011'B: NOT answer_now, new_round
- EM = '0100'B to '1111'B: reserved.

The next 4 bits are the Event_Type, encoded as:

- ET = '0000'B: high priority Event_Round (high_priority)
- ET = '0001'B: low priority Event_Round (low_priority)
- ET = '0010'B to '1111'B: reserved.

The last 4 bits are reserved.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	1	0	0	QT	NR	0	0	0	HP	0	0	0	0
F_code = 9				Event_Mode				Event_Type				reserved			

Figure 95 - General_Event_Request frame format

NOTE General_Event_Request is the only Master Frame without an address, since it is by default broadcast to all devices.

3.6.3.4.2 Group_Event_Request

The master shall poll a group of devices belonging to a group {M, C} by a Group_Event_Request consisting of three fields, as shown in Figure 96:

- The bits 0 to 3 are the F_code = 13;
- The next bits, starting with bit 4, consisting of a set of contiguous "1" closed by a "0" build the mask. The location of the "0" bit depends on the step in the search tree. The number of "1" is equal to M-1;
- The remaining low order bits to the right of the zero until bit 15 define the group address C.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	1	1	1	1	1	0	A	B	C	D	E	F
F_code = 13				mask (don't care)						Group_Address					

Figure 96 - Group_Event_Request frame (M = 6, C = ABCDEF)

EXAMPLE - Assuming all devices have an event pending, if the address is:

1111 1111 1110, all devices respond (= General_Event_Request {NOT new_round, answer_now});

1111 1111 1101, all devices with an odd address respond;

1111 1001 1001, all devices with an address ending in 01 1001 respond;

0111 0110 1000, two devices which differ in their most significant bit respond.

In the above Figure 96, the Group is {M = 6, C = ABCDEF} and it addresses 64 devices.

3.6.3.4.3 Single_Event_Request

The master shall poll a single device for its event by an Single_Event_Request, which is a Master Frame with the following structure, as shown in Figure 97:

- The bits 0 to 3 are the F_code = 14;
- The next 12 bits define an individual device address.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	A	A	A	A	A	A	A	A	A	A	A	A
F_code = 14				Device Address											

Figure 97 - Single_Event_Request frame

3.6.3.4.4 Event_Identifier_Response

A slave with a pending event of the priority established at the beginning of the Event_Round shall respond with an Event_Identifier_Response to:

- a General_Event_Request {answer_now};
- a Group_Event_Request with the group address matching its own; or
- a Single_Event_Request whose address matches the 12-bit Device Address;

The Event_Identifier_Response shall be a 16-bit Slave Frame, the content of which are interpreted as a 4-bit F_code followed by a 12-bit address, as shown in Figure 98.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_code				logical or physical address											

Figure 98 - Event_Identifier_Response frame

A slave shall not use the following F_codes in its Event_Identifier_Response:

- F_code = 8 (Mastership_Transfer);
- F_code = 9 (General_Event_Request);
- F_code = 13 (Group_Event_Request);
- F_code = 14 (Single_Event_Request).

3.6.3.4.5 Event_Read_Request

The master shall re-send a correctly received Event_Identifier_Response as an Event_Read_Request .

The Event_Read_Request shall have the same format as the Event_Identifier_Response, except that it has a Master Start Delimiter instead of the Slave Start Delimiter.

3.6.3.4.6 Event_Read_Response

The format of the Event_Read_Response is that of any Slave Frame.

NOTE The Event_Read_Response will most likely be a Message Data frame.

3.6.4 Devices_Scan

3.6.4.1 Device_Status

All devices shall implement a source port for their Device_Status, containing a 16-bit word which indicates their status, and which can be polled.

3.6.4.1.1 Device_Status_Request

The master shall poll the Device_Status of a device with a Device_Status_Request, which is a Supervisory_Data_Request with F_code = 15, and which contains the 12-bit address of the device, as Figure 99 shows.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	A	A	A	A	A	A	A	A	A	A	A	A
F_code = 15				Device Address											

Figure 99 - Device_Status_Request

3.6.4.1.2 Device_Status_Response

3.6.4.1.2.1 General format

A slave shall respond to a Device_Status_Request with a Device_Status_Response, which is a Supervisory_Data_Response consisting of the following fields as shown in Figure 100.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SP	BA	GW	MD	T	T	T	T	LAT	RLD	SSD	SDD	ERD	FRC	DNR	SER
capabilities				class_specific				common_flags							

Figure 100 - Device_Status_Response**3.6.4.1.2.2 Capabilities field**

The first four bits shall define the basic capabilities of a device:

- SP bit 0 set to "0" indicates the Device_Status and Process Data capability, bit 0 set to "1" indicates a special device, the next 3 bits are not specified;
- BA bit 1 set to "1" indicates the Bus Administrator capability;
- GW bit 2 set to "1" indicates the Gateway capability;
- MD bit 3 set to "1" indicates the Message Data capability.

3.6.4.1.2.3 Specific field

This 4-bit specific field depends on the capabilities field. It distinguishes four specific fields:

- Class 1 device;
- Device with Message Data capability;
- Device with Bus Administrator capability;
- Device with Gateway capability.

3.6.4.1.2.3.1 Class 1 device specific field

A device of Class 1 shall respond with the "capabilities" field set to "0000", as shown in Figure 101.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	LAT	RLD	SSD	SDD	ERD	FRC	DNR	SER
capabilities				class_specific				common_flags							

Figure 101 - Device_Status of Class 1 device

The specific field shall be "0000".

Two bits of the Device_Status word are fixed:

- SER = 0 (no reservation for service possible);
- FRC = 0 (no forcing possible).

3.6.4.1.2.3.2 Message Data specific field

A device capable of Message Data communication according to the Real-Time Protocols of the TCN shall respond with Bit 3 set to "1", as shown in Figure 102.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x	x	1	x	X	x	x	LAT	RLD	SSD	SDD	ERD	FRC	DNR	SER
capabilities				class_specific				common_flags							

Figure 102 - Device_Status of Class 2/3/4/5 device

3.6.4.1.2.3.3 Bus Administrator specific field

A device with the Bus Administrator capability shall respond with Bit 1 set to "1", and set the four bits in the "class_specific" field, as shown in Figure 103:

- AX1: Bit 4 second least significant bit of the Actualisation_Key of the Periodic List;
- AX0: Bit 5 least significant bit of the Actualisation_Key of the Periodic List;
- ACT: Bit 6 is set if the device is actualised and in condition of becoming master;
- MAS: Bit 7 is set if the device is the current master.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	x	x	AX1	AX0	ACT	MAS	LAT	RLD	SSD	SDD	ERD	FRC	DNR	SER
capabilities				Class_specific				common_flags							

Figure 103 - Device_Status of a device with Bus Administrator capability

3.6.4.1.2.3.4 Gateway specific field

A device with the Gateway capability shall respond with its Bit 2 set to "1".

If the device has at the same time the Bus Administrator capability, the "specific" field shall be that of a Bus Administrator, as shown in Figure 104.

If the device is a gateway without Bus Administrator capability, the device shall set the "class_specific" field as shown in Figure 103:

- STD: Bit 4 set to "1" indicates a static disturbance (remote bus down);
- DYD: Bit 5 set to "1" indicates a dynamic disturbance (e.g. train inauguration);
- RV6 Bit 6 is "0" (reserved); and
- RV7 Bit 7 is "0" (reserved).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	x	1	x	STD	DYD	RV6	RV7	LAT	RLD	SSD	SDD	ERD	FRC	DNR	SER
capabilities				class_specific				common_flags							

Figure 104 - Device_Status of a device with Gateway capability

NOTE The STD and DYD bits can be read directly on a gateway which has the Bus Administrator capability.

3.6.4.1.2.4 Common field

These eight bits are common to all devices:

LAT: Line_A_Trusted:

shall be asserted if the Master Frame of this telegram was received over Line_A, negated if it was received over Line_B;

RLD: Redundant_Line_Disturbed:

shall be set if the Observed_Line is disturbed (3.3.2.5);

SSD: Some_System_Disturbance:

shall be set when the sink time supervision of any Port triggered and reset when all configured Ports operated normally;

SDD: Some_Device_Disturbance:

shall be set by a device malfunction (for instance: ROM or RAM checksum error) or a fault outside of the device (for instance: damaged sensors, ...) and reset when the fault is removed;

ERD: Extended_Reply_Delay:

shall be asserted if $T_{\text{ignore}} > T_{\text{reply\_def}}$, negated otherwise; (3.4.2.4.2);

FRC: forced device:

shall be asserted if any Port has been forced to an imposed value and negated if all Ports attached to the MVB are in the unforced state;

DNR: Device_Not_Ready:

shall be asserted if the device is not operational (for instance application not running) but able to operate normally on the bus, and negated if the device is operational;

SER: System Reserved:

shall be set if the device has been reserved for exclusive use and reset when this exclusive use is lifted or timed-out.

3.6.4.2 Device_Status protocol

When receiving a Master Frame with $F\_code = 15$ (Device_Status_Request) and with its own address, a device shall reply with its Device_Status_Response.

NOTE 1 The destination of the Device_Status frames is normally the master, but it can also be a monitoring device which operates by invitation.

NOTE 2 Reading the Device_Status may cause clearing of the RLD bit in Class 1 devices and some other devices, see 3.3.2.5.

3.6.4.3 Devices_Scan protocol

NOTE A configured master holds a list of the devices configured by Network_Management, the Known_Devices_List. The list of found devices, the Devices_List may differ from the Known_Devices_List because of device failure or configuration error.

To supervise the bus, the master shall regularly poll the Device_Status of all devices and build the Devices_List, which contains the address and Device_Status of each found device.

The master shall list the devices with Bus Administrator capability it finds in the Bus Administrators_List.

The master shall scan the known devices in its Known_Devices_List, and also unknown devices, to check for insertion of new devices.

The scan rate for known and unknown devices is a configuration parameter.

It is recommended that the Devices_Scan be performed at the rate of 64 devices every 512,0 ms.

A device which ceases to respond for three consecutive polls shall be removed from the Devices_List.

3.7 Mastership transfer

Mastership Transfer describes the protocols which select a master from one of several bus administrators, and ensures mastership transfer at the end of a turn or upon the occurrence of a failure.

3.7.1 Mastership transfer operation

3.7.1.1 Bus administrator configuration

3.7.1.1.1 Bus_Configuration

The information which a bus administrator needs for regular operation (T_reply, Periodic List, etc.), shall be contained in the Bus_Configuration data structure.

The Bus_Configuration shall be identified by an Actualisation_Index, which identifies its version and which shall be incremented by one for each new version loaded.

The Bus_Configuration may be in addition identified by a 15-bit Actualisation_Key, which identifies its version of the configuration and which (if used) shall be incremented by one for each new version loaded into the bus administrator. The two lowest significant bits of the Actualisation_Key may be used as Actualisation_Index.

A bus administrator in possession of a Bus_Configuration (3.8.4.6) is said to be "configured". It shall set the ACT bit in its Device_Status to "1".

An "unconfigured" bus administrator only executes the Event_Round and the Devices_Scan. It shall set the ACT bit in its Device_Status to "0".

A master shall assume that another bus administrator is "actualised" if its Actualisation_Index is one higher or equal to its own.

A bus administrator shall revert to the "unconfigured" state if the Actualisation_Index has been modified four times in sequence, but the bus administrator has not become master.

NOTE The generation of the Actualisation_Key is at charge of Network Management.

3.7.1.1.2 Bus Administrators_List

Each bus administrator shall hold a circular list of all other bus administrators, called the Bus Administrators_List, ordered as the token circulates, and preferably structured in ascending order of the device addresses.

NOTE The Bus Administrators_List may be part of the Bus_Configuration, or may be generated on-line by the Devices_Scan.

3.7.2 Mastership transfer specifications

The bus administrator states are described in the SDL syntax in Figure 105.

3.7.2.1 States

3.7.2.1.1 State "STANDBY_MASTER"

In this state, the bus administrator does not exercise mastership. It shall go to the state "REGULAR_MASTER":

- if it receives no Master Frames during a time-out T_{standby} , or
- if it receives a mastership offer from the current master to which it can respond (i.e. the bus administrator is "actualised").

3.7.2.1.2 State "REGULAR_MASTER"

In this state, the bus administrator shall exercise mastership (it execute the Periodic Polling, Event Polling and Devices_Scan).

If it detects a Master Frame which it did not send itself, the bus administrator shall assume a master collision, and return to the state "STANDBY_MASTER".

After its turn elapses or after a "resign" command, the bus administrator shall go to the state "FIND_NEXT".

3.7.2.1.3 State "FIND_NEXT"

In this state, the bus administrator shall fetch the address of the proposed master in its Bus Administrators_List and poll its Device_Status.

The proposed master is a valid bus administrator if its Device_Status states that:

- a) it has the Bus Administrator capability ($BA = 1$);
- b) it is configured ($ACT = 1$);
- c) it has an Actualisation_Index equal to or higher by 1 than that of the current master, (Modulo 4).

The bus administrator shall send a Mastership_Transfer_Request to itself and then to the valid bus administrator and go to the state "INTERIM_MASTER".

Otherwise, the bus administrator shall return to the state "REGULAR_MASTER" and it shall try the next bus administrator in its Bus Administrators_List at the next turn.

3.7.2.1.4 State "INTERIM_MASTER"

In this state, the bus administrator shall expect the Mastership_Transfer_Response from the next bus administrator.

When it receives an (accept or reject) response or when the time-out elapses, the bus administrator shall go the state "STANDBY_MASTER". If the response of the other bus administrator is "reject", the bus administrator shall report an error.

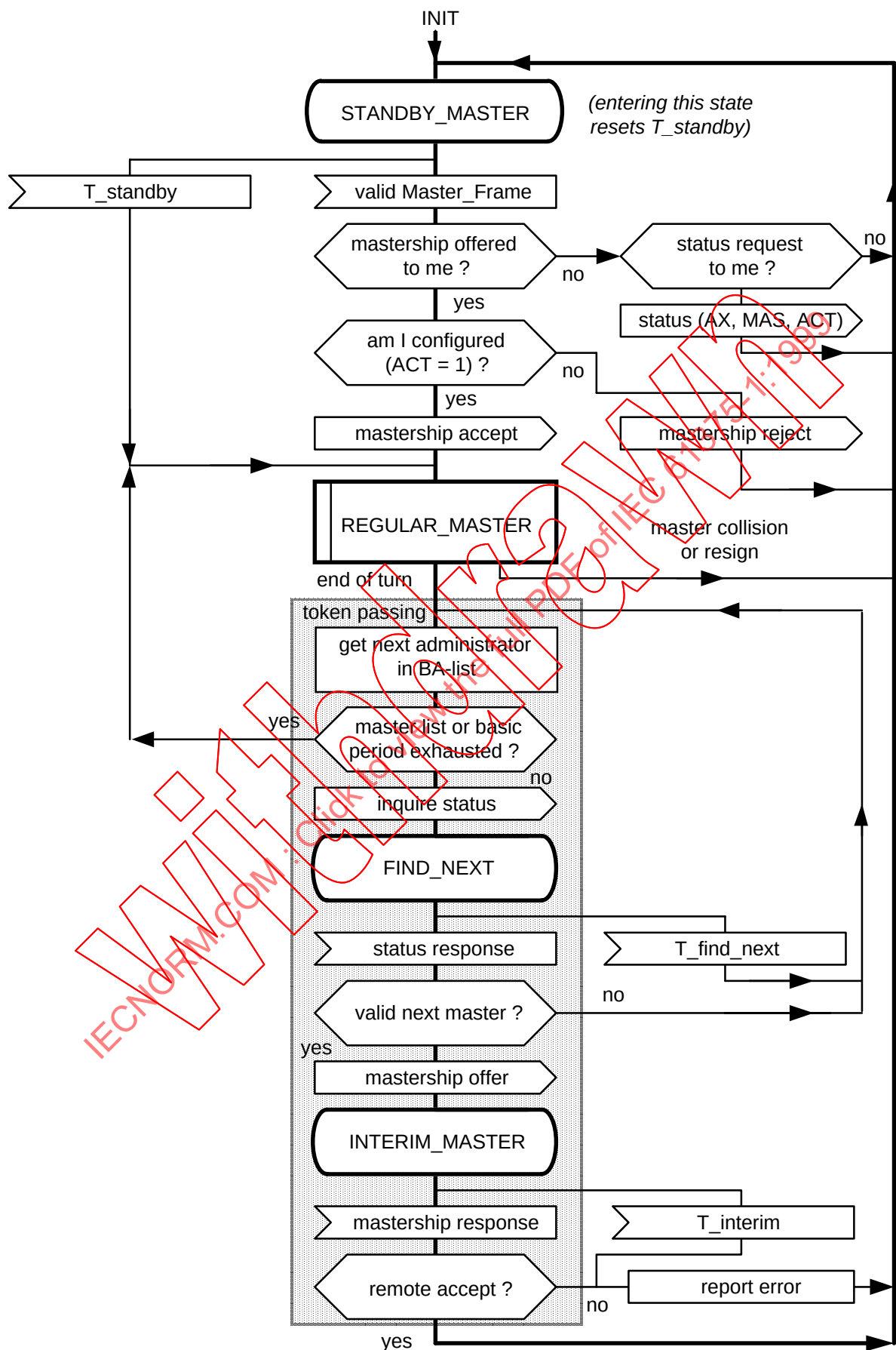


Figure 105 - Mastership Transfer states

3.7.2.2 Time-outs for mastership transfer

3.7.2.2.1 Turn duration

A turn shall last a multiple of 1024 ms and shall not exceed 256×1024 ms.

3.7.2.2.2 Time-outs of "STANDBY_MASTER"

T_standby shall be equal to:

- $(T_{\text{alive}} \times 2 \times (1 + \text{rank\_in\_ba\_list}))$ for a configured bus administrator; or
- $(T_{\text{alive}} \times 2 \times (\text{ba\_adr} + 15))$ for an unconfigured bus administrator; or
- infinite for a disabled bus administrator.

where:

T_alive is the maximum interval between Master Frames;

rank_in_ba_list is the order of that device in the Bus Administrators_List;

ba_adr is the Device Address of the bus administrator.

NOTE Allocating the lower Device Addresses to the bus administrators speeds up recovery.

3.7.2.2.3 Time-outs of "FIND_NEXT"

T_find_next shall be equal to T_reply

3.7.2.2.4 Time-outs of "INTERIM_MASTER"

T_interim shall be equal to T_reply.

3.7.2.3 Supervisory Data frames for mastership transfer

3.7.2.3.1 Device Status telegram

A master shall check if the proposed master is active and properly configured by reading the Device_Status of the proposed master through a Device_Status_Request, as shown in Figure 106.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_code = 15				Device Address of proposed master											

Figure 106 - Device_Status_Request (sent by current master)

The proposed master shall respond with a Device_Status_Response, as shown in Figure 107.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	1	x	x	AX1	AX0	ACT	MAS	LAT	RLD	SSD	SDD	ERD	FRC	DNR	SER
capabilities				class_specific				common_flags							

Figure 107 - Device_Status_Response (sent by proposed master)

3.7.2.3.2 Mastership transfer telegram

The current master shall send a Mastership_Transfer_Request containing, as shown in Figure 108:

- the F_code = 8;
- the address of the proposed master or its own address (self-transfer).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F_code = 8				Device Address of proposed master											

Figure 108 - Mastership_Transfer_Request (sent by current master)

The master or proposed master shall respond with a Mastership_Transfer_Response, which is a Supervisory_Data_Response consisting, as shown in Figure 109, of:

- a) Bit 0 = ACP,
 - if ACP = 1, indicates that the bus administrator accepts mastership;
 - if ACP = 0, indicates that the bus administrator rejects mastership;
- b) Bit 1 to Bit 15 contain the Actualisation_Key of the proposed master (or 0 if it is not used).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ACP	Actualisation_Key														

Figure 109 - Mastership_Transfer_Response (sent by proposed next master)

3.8 Link Layer Interface

3.8.1 Link Layer layering

The Link Layer Interface of the MVB shall provide three services as shown in Figure 110:

- a) The Link Process Data Interface (LPI), which provides a set of Ports in which Process Data are stored. This interface is specified in Clause 2 and only the parameters specific to the MVB are specified in this standard;
- b) The Link Message Data Interface (LMI), which provides a set of queues for Message Data, used by the upper layers (network, transport, session). This interface is specified in Clause 2 and only the parameters specific to the MVB are specified in this standard;
- c) The Link Supervision Interface (LSI), which provides the services for the configuration of the link layer and for the supervision of the bus. This interface is specific to the MVB and is specified in 3.8.4.

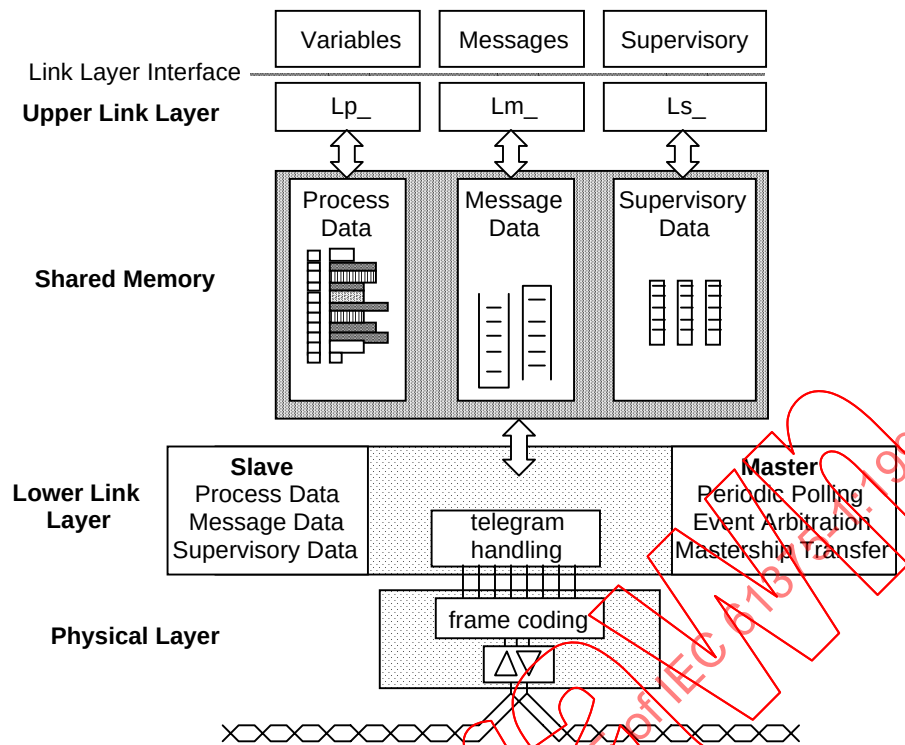


Figure 110 - Link Layer Layering

3.8.2 Link Process Data interface

The Process Data interface is specified in Clause 2. The following defines parameters particular to the MVB.

The length of a Dataset shall be limited to 256 bits.

The Port_Address of a dataset shall be 12-bit long. It shall be the same as the address used to transmit the dataset over the bus.

The number of Datasets per Traffic_Store shall be limited to 4095.

The dataset with address 4095 shall be reserved for synchronisation and time distribution.

Where exact synchronisation is required, the end of the Master Frame shall serve as a reference point.

NOTE A Dataset is a group of variables transmitted in the same Process Data frame.

3.8.3 Link Message Data interface

3.8.3.1 General

The Link Message Data Interface provides services for sending Message Data frames and services for retrieving received Message Data.

The Link Message Data Interface provides the basic service on which the message services are built.

The Link Message Data Interface (LMI) is specified in Clause 2. The following specifies which parameters are particular of the MVB.

This interface is not implemented in devices lacking the Message Data capability.

3.8.3.2 Priority

The Link Message Data Interface builds on an output queue, which may be attached to either of the two priority levels (ET = 0 and ET = 1) of the sporadic medium access, and on an input queue, which is not prioritised.

The allocation of the priorities is application dependent.

3.8.3.3 Packet size

The "SZ" field in a void packet shall be set to zero.

The maximum length the "SZ" field shall indicate is 27 (32 octets - 5 octets of link header, including 1 octet for the "SZ" field).

3.8.3.4 Protocol_Type

The value of the Protocol_Type for Real-Time Protocols shall be '1000'B. Other values for the Protocol_Type are reserved for future use.

3.8.3.5 Message Transport Protocol

An MVB device shall announce a packet size of 27 octets when opening a connection.

When responding to a connection opening, an MVB device shall specify a packet size equal to 27 octets.

3.8.4 Link Supervision Interface

3.8.4.1 General

The Link Supervision Interface is specific to the MVB.

It provides general services for configuration and inspection of the Link Layer and event reporting.

The Link Supervision Interface is grouped into 6 services:

- a) MVB_Status: expresses the state of the link layer;
- b) MVB_Control: controls the operation of the device;
- c) MVB_Devices: provides the Device_Status of the other devices on the same bus;
- d) MVB_Administrator: sets up the bus administrator;
- e) MVB_Report: event reporting.

NOTE The access to these objects may be through interface procedures or may be hard-wired. Whichever the method, a device shall be able to provide the functionality described.

The procedures of this interface are prefixed by ls_v_ (link supervision, vehicle bus).

The format of the parameters of the following interface procedures is not indicated. However, the Train Network_Management document (Clause 5) proposes a format which is recommended as a parameter format.

3.8.4.2 Link Supervision Interface procedures

The services are defined by data structures which can be read and written with a procedure of the following form:

Definition	MVB link layer access procedure
Syntax	LS_RESULT ls_v_<service_name> (<parameters>)

The possible results are listed in Table 54.

Table 54 - LS_RESULT encoding

Constant	Code	Meaning
LS_OK	0	success
LS_BUSY	1	try again later
LS_CALL_SEQUENCE	2	wrong command sequence

Only the parameters of the procedures are defined here.

3.8.4.3 MVB_Status

The MVB_Status object describes the static and dynamic parameters of the link layer. This structure is intended to be transmitted without modifications.

Every device shall have an MVB_Status object with the attributes listed in Table 55.

Table 55 - MVB_Status object

Attribute	Type	Meaning
device_address	WORD16	address of the device on the bus. Most significant four bits = '0000'B.
mbv_hardware_name	STRING32	identifies the MVB controller, the hardware and its version.
mbv_software_name	STRING32	identifies the MVB software version, preferably in the format: xxxx-Vz.z-dd.mm.yy.
device_status	Device_Status	copy of the 16-bit Device_Status.
reserved	WORD8 (=0)	used for alignment.
t_ignore	UNSIGNED8	configured time-out value for the delay between a Master Frame and a Slave Frame on that device in multiples of 1 μ s. if t_ignore = 0, the value is the default value T_reply_def = 42,7 μ s.
lineA_errors	UNSIGNED32	32-bit counter incremented each time the Line_Disturbed signal of Line_A is set; this counter does not wrap-around when reaching its highest value.
lineB_errors	UNSIGNED32	32-bit counter incremented each time the Line_Disturbed signal of Line_B is set; this counter does not wrap-around when reaching its highest value.

3.8.4.4 MVB_Control

The MVB_Control object controls the static and dynamic parameters of the link layer. This structure is intended to be transmitted without modifications.

Every device shall have an MVB_Control object including the parameters listed in Table 56.

Table 56 - MVB_Control object

Attribute	Type	Meaning
device_address	WORD16	address of device on the bus, most significant four bits = 0 (if software-modifiable).
t_ignore	UNSIGNED16	value of the time-out T_ignore to be used, in multiples of 1 μ s default (t_ignore = 0): T_reply_def = 42,7 μ s recommended values: 22,0 μ s, 64,0 μ s and 86,0 μ s maximum value: 255,0 μ s.
command	BITSET8	command to be performed by the MVB Link Layer:
aon	(0)	(Bus Administrator only) enables the bus administrator
aof	(1)	(Bus Administrator only) disables the bus administrator
spl	(2)	(Bus Administrator only) switch to new Periodic List (= 1). Causes the bus administrator to switch to the new Periodic List next time the bus administrator becomes master
tms	(3)	(Bus Administrator only) transfer mastership to the next bus administrator in the Bus Administrators_List at the end of the macro-cycle following the request, retain mastership if other bus administrator does not accept it.
sla	(4)	trust always Line_A, even if this line is disturbed (sla = slb = 0: pas d'action)
slb	(5)	trust always Line_B, even if this line is disturbed (sla = slb = 1: normal redundant operation)
cla	(6)	reset frame error counter of Line_A (cla = clb = 0: no action)
clb	(7)	reset frame error counter of Line_B (cla = clb = 1: reset both counters)

NOTE On devices which are not bus administrators, redundant lines are switched when the bus administrator reads the Device_Status.

3.8.4.5 MVB_Devices

The MVB_Devices object contains in particular the list of the devices and their Device_Status ordered by increasing address, indicating for each:

- the Device Address;
- the Device_Status read during the latest Devices_Scan.

Every bus administrator shall have an MVB_Devices object with the attributes listed in Table 57.

Table 57 - MVB_Devices object

Attribute	Type	Meaning
nr_devices	UNSIGNED16	number of devices in the Devices_List.
devices_list	ARRAY [nr_devices] OF	list of the found devices, comprising for each:
device_address	WORD16	12-bit address of the device (four most significant bits = 0)
device_status	Device_Status	copy of the 16-bit Device Status of the device.

3.8.4.6 MVB_Administrator

The MVB_Administrator object contains in particular:

- the configuration parameters (Basic Period, T_reply, Event strategy, etc.);
- the Known_Devices_List and the Devices_Scan algorithm;
- the Periodic List, consisting of the Cycle_Lists and the Split_Lists;
- the Bus Administrators_List.

Every bus administrator shall have an MVB_Administrator object with the attributes listed in Table 58.

Table 58 - MVB_Administrator object

Attribute	Type	Meaning
checksum0	UNSIGNED16	Checksum over this structure - application dependent. This word is located at offset 0 of this structure.
actualisation_key	UNSIGNED16	Version number of the configuration, as a 16-bit number. The most significant bit set to 1 indicates to transmit the least significant 15 bits in Mastership_Transfer_Response. The two least significant bits are copied to AX0 and AX1.
t_reply_max	UNSIGNED16	Maximum expected reply delay between a Master Frame and a Slave Frame, in multiples of 1µs default (t_reply = 0) = 42,7 µs, maximum = 255,0 µs. In case of Extended Reply Delay operation, this value shall be equal or greater to the largest value of the T_ignore time-out of any device in this configuration.
macro_cycles	UNSIGNED16	Number of basic periods after which the periodic traffic repeats itself (e.g. 512 for a 2,0 ms T_bp).
event_poll_strategy	ENUM16	'0000'H no event polling '4000'H high priority event polling only '8000'H low priority event polling only 'C000'H high and low priority event polling.
basic_period	UNSIGNED16	T_bp in microseconds (recommended values: 1000, 1250, 1042, 2000, 2043, 2500).
macrocycles_per_turn	UNSIGNED16	Number of Macro_Cycles after which the master shall offer mastership to the next bus administrator in its list.
devices_scan_strategy	ENUM16	0: scan all known devices addresses 1: scan all device addresses
reserved2	WORD16	unused
reserved3	WORD16	unused
reserved4	WORD16	unused
reserved5	WORD16	unused
known_devices_list_offset (offset_KDL)	UNSIGNED16	Offset (from checksum0) where the first address of the device list is located. The list size is equal to offset_RSL - offset_KDL.
reserved_list_offset (offset_RSL)	UNSIGNED16	unused (offset_RSL = offset_PL)
periodic_list_offset (offset_PDL)	UNSIGNED16	offset (from checksum0) where the first F_code and address of the first cycle is located. The list size is equal to offset_BAL - offset_PDL.

Table 58 (concluded)

Attributes		Type	Meaning
bus_administrators_list_offset (offset_BAL)		UNSIGNED16	offset (from checkword0) where the first address of the Bus Administrators_List is located.
devices_scan_list_offset (offset_DSL)		UNSIGNED16	offset (from checkword0), where the device scan allocation list is located (unused)
end_list_offset (offset_END)		UNSIGNED16	dummy offset indicating the size of the devices_scan_list and pointing to the next location after the last device of that list.
known_devices_list		ARRAY [nr_devices] OF	list of the known devices, (nr_devices = offset_KDL - offset_RSL), comprising for each:
	device_address	WORD16	12-bit address of the device (four most significant bits =0)
cycle_lists_offsets		ARRAY [11] OF	array of 11 elements, comprising for each one of the possible 11 cycles:
	cycle_n_offset	UNSIGNED16	offset (from beginning of Periodic List) where the first address of the cycle is located
split_list_offsets		ARRAY [5] OF	array of 5 elements, comprising for each:
	split_n_m_offset	UNSIGNED16	offset (from beginning of Periodic List) where the first address of the table split_n_m is located
cycle_lists		ARRAY[11] OF	array of 11 lists, each comprising
	cycle_n	ARRAY [cycle_size] OF	array of [cycle_size] values, (cycle_size is the offset between this list and the next) each comprising:
	f_code	ENUM4	F_code (any value) of the Master Frame to be sent
	address	UNSIGNED12	address in the Master Frame
split_lists		ARRAY [5] OF	array of five lists (n= 4, n=16, n=64, n= 256, n= 1024) and m = n/2, each comprising:
	split_n_m	ARRAY [n] OF	array of [n] values, each comprising:
	lower_count (n)	UNSIGNED8	number of frames of the lower cycle (n) to be put into each Basic Period in the most significant octet (this list is repeated twice; it is only half as long as the upper_count list).
	upper_count (m)	UNSIGNED8	number of frames of the upper cycle (m) to be put into each Basic Period
bus_administrators_list		ARRAY [nr_admins] OF	list of the known bus administrators, (nr_admins = devices_scan_list_offset - bus_administrators_list_offset), comprising for each:
	f_code	UNSIGNED4	F_code = 8 (ready for mastership transfer)
	device_address	UNSIGNED12	12-bit address of the device.

3.8.4.7 MVB_Report

Definition	Reports are given to the application by calling a subscribed function of the type:
Syntax	<pre>typedef void (* ls_v_report) (ls_report)</pre>

The value of the reports shall be as given as encoded in Table 59.

Table 59 - LS_V_REPORT encoding

Constant	Code	Meaning
LS_V_FRAME_COUNT	1	number of received frames reached 65526.
LS_V_HW_ERROR	2	some disturbance occurred
LS_V_CHANGE	3	(Bus Administrator only) change in devices list
LS_V_REPEATED_EVENT	4	(Bus Administrator only) a device keeps on signalling the same event
LS_V_SYNCHRO	5	synchronisation received
LS_V_MASTER_CHANGE	6	this device transferred mastership.

4 Wire Train Bus (WTB)

This clause applies to Open Trains; it can also be used after agreement between user and manufacturer in other applications.

A tutorial on the WTB is given in Clause A.4.

4.1 General

4.1.1 Contents of this clause

This clause specifies one component of the Train Communication Network, the Wire Train Bus (WTB), a serial data communication bus designed primarily, but not exclusively, for interconnecting vehicles which are frequently coupled and uncoupled, as is the case of international UIC trains.

Figure 111 illustrates the WTB application.

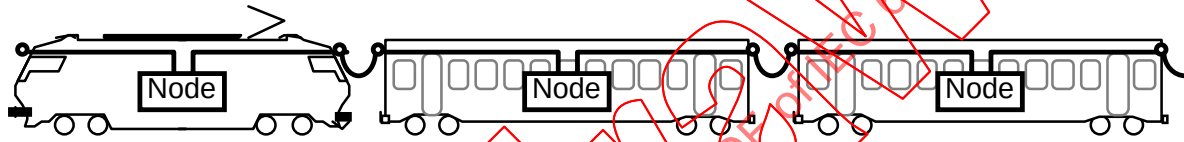


Figure 111 - Wire Train Bus

This clause specifies:

- the physical medium in single-line and double-line configurations;
- the signalling and the redundancy handling;
- the format and timing of the transmitted frame and telegrams;
- the organisation of the bus traffic;
- the allocation of mastership and the naming of the nodes (inauguration);
- the recovery procedure in case of node or line failure;
- the link layer interface and the layer management interface.

4.1.2 Structure of this clause

This standard follows the layered structure of a reference WTB node, as shown in Figure 112:

Subclause 4.1 General

- normative requirements and definitions

Subclause 4.2 Physical Layer

- cable and connection to the nodes, node structure
- signal levels and signal timing specifications

Subclause 4.3 Medium-dependent signalling

- frame encoding and decoding
- physical redundancy handling

- Subclause 4.4 Frames and telegrams

- frame format
- telegram format
- telegram timing

Subclause 4.5 Link Layer Control

- addressing
- telegram types

Subclause 4.6 Medium Allocation

- periodic traffic
- sporadic traffic
- supervisory traffic

Subclause 4.7 Inauguration

- Master selection
- Node naming
- Topography distribution
- Mastership redundancy

Subclause 4.8 Link layer interface

- Process Data
- Message Data
- Layer Management Interface

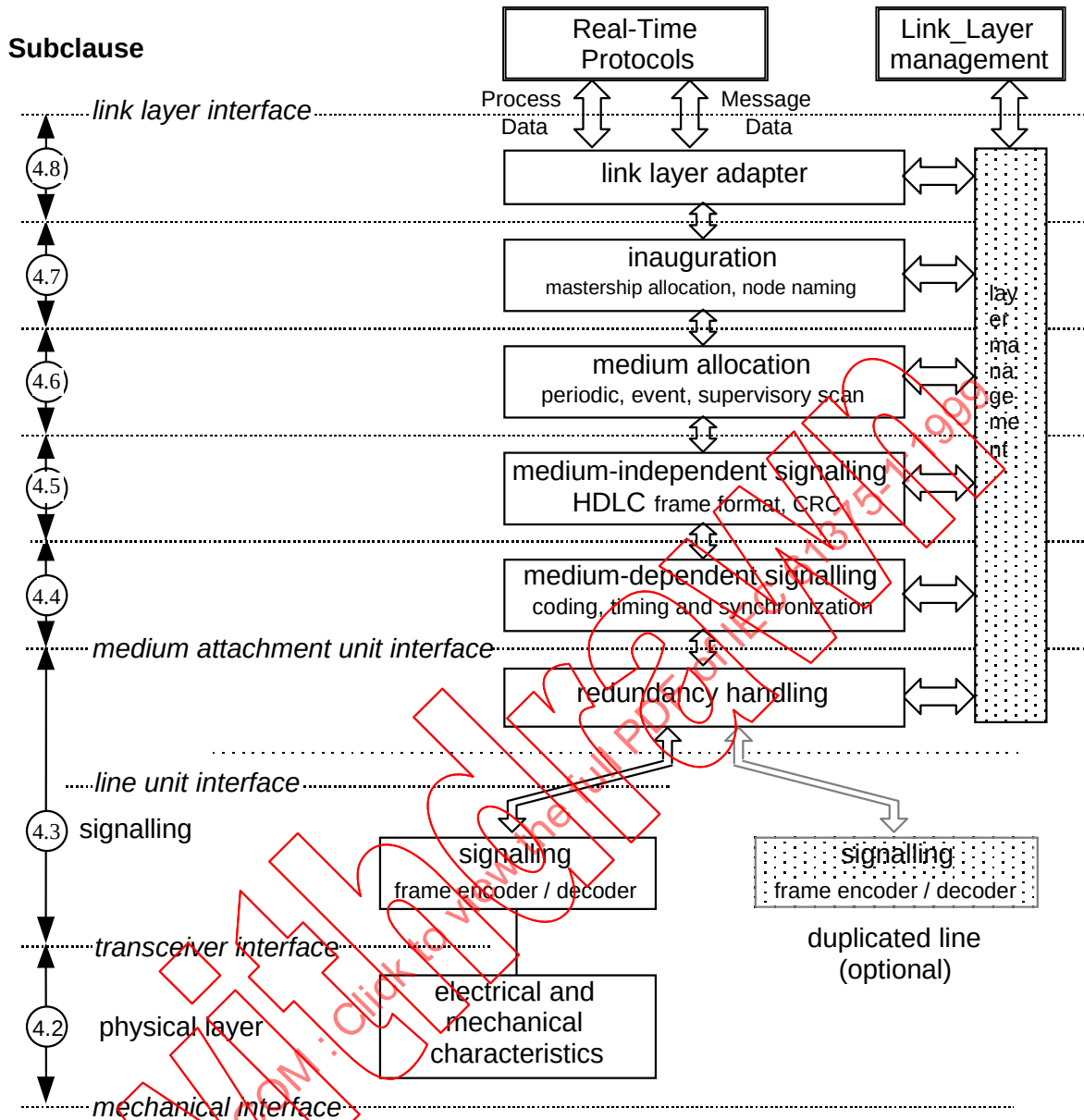


Figure 112 - Reference model of the WTB

4.2 Physical layer

This subclause specifies the physical medium of the WTB as a shielded, twisted wire pair bus operated at 1,0 Mbit/s.

This standard intends that the different sections, nodes and connectors provide an electrical medium as uniform as possible in respect of signal propagation.

4.2.1 Topology

4.2.1.1 Bus sections

The WTB bus shall consist of nodes interconnected by bus sections of the following types:

- a) trunk cables running along a vehicle (continuity vehicles only have a trunk cable);
- b) jumper cables connecting the trunk cables of different vehicles;
- c) extension cables extending the trunk cable to reach a Node.

4.2.1.2 Couplers

Connectors and junction boxes may be used to assemble nodes and cable sections.

Each vehicle carries a portion of the bus and a number of nodes.

4.2.1.3 Nodes

In regular operation, each Node shall be inserted into the trunk cable and connected to two bus sections:

- a) nodes located at the ends of the bus, or End Nodes, shall electrically terminate the two bus sections attached to them;
- b) nodes located in the middle of the bus, or Intermediate Nodes, shall electrically connect the two bus sections attached to them.

The two cable sections attached to a node shall be named Direction_1 and Direction_2.

There may be one or several nodes per vehicle in a train, as shown in Figure 113.

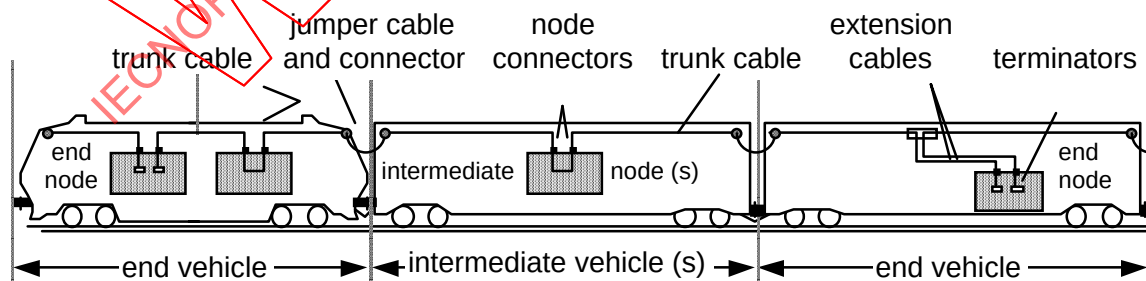


Figure 113 - Train Composition (two Intermediate Nodes shown)

4.2.1.4 Vehicle orientation

Where orientation of the nodes is critical to the left-right recognition, the following conventions shall be observed:

- a) One end of the vehicle is identified as Extremity 1, the other as Extremity 2;
- b) Direction_1 of a node is attached towards Extremity 1, and Direction_2 is attached towards Extremity 2;
- c) If Direction_1 points north, the side of the vehicle that points west is named side A, the side which points east is named side B;
- d) A node uses the same conventions for A and B as the vehicle it is located in.

NOTE 1 Direction_1 of one node may point either to Direction_1 or Direction_2 of another node, except if both nodes are in the same vehicle, since the orientation of vehicles with respect to one another is not predictable.

NOTE 2 DIN 25005 defines Extremity 1 of a vehicle as the end of the vehicle opposite to where the parking brake is located.

4.2.1.5 Vehicle specification (informal)

Since a manufacturer may supply individual vehicles or nodes rather than a complete train, the specifications of the bus, the vehicle and the nodes are treated separately.

This standard specifies the characteristics of the whole bus and of each node. For a particular application, the vehicle characteristics are deduced from them. The conformance of the train is then met providing the number of vehicles or nodes is not exceeded.

The following calculations, used in UIC leaflet 557, apply to railway vehicles as specified in the UIC 556 leaflet. Other applications, such as mass transit, may use a similar calculation.

The reference train composition consists of 22 vehicles, giving a cable length of 860,0 m without the use of repeaters. There is normally only one node per vehicle, but up to 10 vehicles may support a second node, for a total of 32 nodes.

According to 4.2.5.2, a node attenuates the signal by less than 0,3 dB (at the nominal frequency), the total attenuation due to the nodes will not exceed $32 \times 0,3 \text{ dB} = 9,6 \text{ dB}$.

According to 4.2.6.3, receivers can handle a dynamic range of 20,0 dB, the remaining attenuation over the whole train for the cable, connectors and other elements may not exceed $20,0 - 9,6 = 10,4 \text{ dB}$.

The maximum attenuation allocated to a vehicle is therefore $10,4 \text{ dB} / 22 = 0,5 \text{ dB}$.

This value is measured with the node(s) removed and their connections short-circuited. The jumper cable is considered in the measurement, as shown in Figure 114.

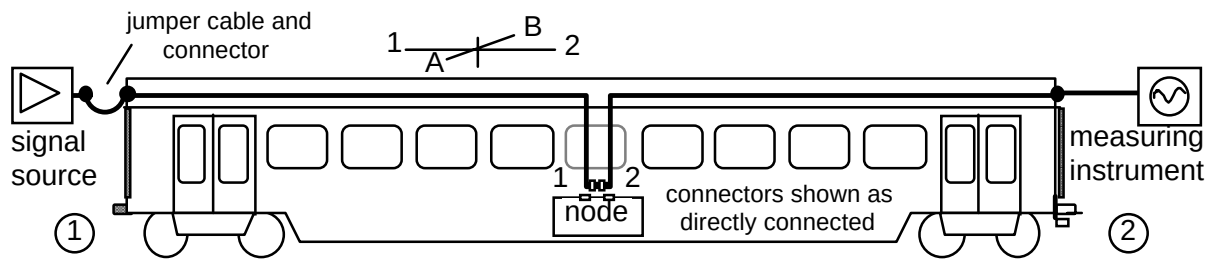


Figure 114 - Vehicle measurement

Due to meanders and extension cables, the cable length per vehicle is about 150% of the vehicle length. Assuming a vehicle length of 26,0 m, the medium shall span a distance in excess of 860 m ($22 \times 26,0 \times 1,5 = 858,0$ m).

To meet these requirements, the medium should present an attenuation of less than 10,4 dB/860,0 m, or 12,0 dB/km. Since jumper cables, connectors and splices may introduce a higher attenuation, a trunk cable with less attenuation than 10,0 dB/km is recommended (4.2.2.4.4).

The same principle is applicable to the other distortion parameters. The measurement scheme is explained in 4.2.5.2.1.

When the vehicle is equipped with redundant Line_A and Line_B, the test applies to each line individually.

4.2.2 Medium specifications

4.2.2.1 Topology

Nodes shall be inserted into the WTB cable, each node being attached to two bus sections, as shown in Figure 115.

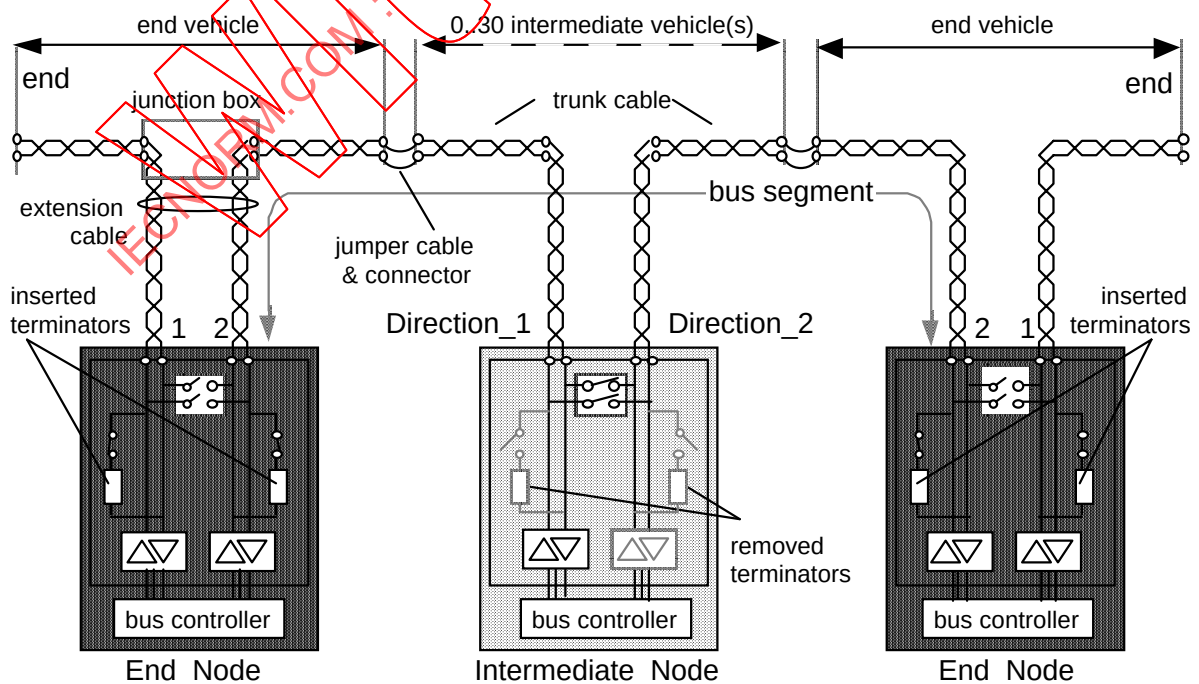


Figure 115 - Connected nodes in regular operation

A node shall be able either:

- to establish electrical continuity between the two bus sections connected to it, to perform as Intermediate Node or
- to terminate electrically the bus sections attached to it through a terminator (impedance adaptation network), to perform as End Node.

End Nodes shall be able to send and receive over both their bus sections independently, while Intermediate Nodes shall have only one of their transceivers enabled.

4.2.2.2 Duplicated medium (option)

This standard defines a redundancy scheme in which each node is attached to two lines, through independent Line Units, as shown in Figure 116.

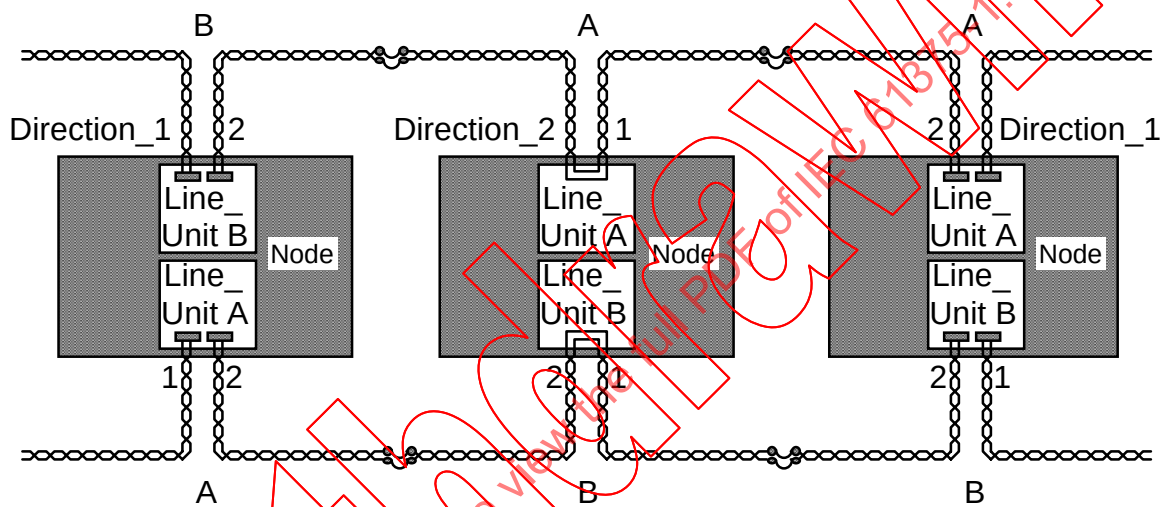


Figure 116 - Double-line attachment

In case this option is used, the following specifications apply:

- The lines shall be identified as Line_A and Line_B;
- This identification shall be consistent for all nodes within the same vehicle;
- Cables belonging to different lines shall be marked distinctly;
- Line_A and Line_B shall be identically configured with respect to Direction_1 and Direction_2.

NOTE 1 For UIC vehicles, the double-line medium is mandatory since it is not possible to connect only one line between vehicles.

NOTE 2 Line_A is associated with side A of the vehicle, and Line_B associated with side B.

NOTE 3 Since the orientation of vehicles is not predictable, Line_A of one vehicle may be connected to Line_B of another vehicle, as shown in Figure 116.

4.2.2.3 Bus Configuration rules

These specifications apply to a bus operating at its maximum foreseeable extent.

Unless otherwise specified, all electrical values shall be measured with a 1,0 MHz $\pm$ 0,01% sinusoidal signal with a differential amplitude of $\pm$ 4,0 V (8,0 Vpp).

4.2.2.3.1 Signalling speed

All bus segments shall operate at the same speed of 1,0 Mbit/s $\pm$ 0,01%, which, due to the Manchester encoding, corresponds to a signalling frequency of 1,0 MHz (BT = 1,0 μ s, BR = 1,0 MHz).

4.2.2.3.2 Delay due to nodes and cabling

T_{pd}, the end-to-end propagation delay on the bus shall not exceed 60,0 μ s between any two nodes.

NOTE 1 The propagation delay for a given application may be evaluated as $T_{pd} = (L \times 6,0 + R \times T_{rd})$, where:

- L is the length of the cable (trunk, extension and jumper cables) in metres;
- 6,0 ns/m approximates the propagation delay of a loaded transmission line;
- R is the number of repeaters; and
- T_{rd} is the propagation delay of a repeater.

NOTE 2 The WTB can bridge 860,0 m without repeater, but repeaters can be useful in some applications.

NOTE 3 This specification matches the tolerable delay as specified in 4.4.2.2 and 4.3.2.2.

4.2.2.3.3 Attenuation due to nodes and cabling

The total voltage attenuation between any two nodes located on the same segment shall not exceed 20,0 dB, measured with a sinusoidal signal at a frequency between 0,5 BR and 2,0 BR.

NOTE 1 The attenuation is proportional to the number of nodes and to the total cable length.

NOTE 2 This specification matches the receiver's dynamic range specified in 4.2.6.3.

4.2.2.3.4 Jitter due to nodes and cabling

A terminated segment at its maximum extension and supporting the maximum number of nodes shall add no more than $\pm$ 0,1 BT of edge jitter, referenced to the idealised zero-crossings;

Test conditions:

- The line is driven by a source of differential amplitude 4,0 V_{pp} $\pm$ 10 %, centred on 0,0 V, through a source impedance of 22,0 Ω $\pm$ 10 %;
- The driving signal is a pseudo-random sequence of '0' and '1' Manchester symbols with a repetition period of at least 511 bits.

NOTE 1 Interference and reflections due to impedance mismatches between the sections, stubs, connectors or load clustering can introduce jitter in the timing of the zero-crossings.

NOTE 2 This specification, taken from ISO/IEC 8802-3, matches the receiver's tolerable jitter specified in 4.2.6.3.

4.2.2.3.5 Skew between redundant lines (option)

The maximum difference in propagation delay between Line_A and Line_B shall not exceed T_{skew} = 30,0 μ s between any two nodes.

NOTE This specification matches the receiver's tolerable skew as specified in 4.3.2.4.1.

4.2.2.4 Cable specification

4.2.2.4.1 Mechanical

All cable sections shall consist of a twin conductor, twisted and shielded, jacketed cable.

The pair shall have no less than 12 twists per metre.

The recommended cross-section of the trunk cable wires is 0,75 mm² (AWG 18).

The recommended cross-section of the jumper cable wires is 1,34 mm² (AWG 16).

If indirect attachment through Sub-D connectors as in 4.2.3.4 is used, the cross-section of each wire of the extension cable shall be no more than 0,56 mm² (AWG 20)

4.2.2.4.2 Marking

The individual wires of the twisted pair shall be identified as X and Y, the shield as S.

The individual wires of the cable shall be marked distinctly.

The marking shall be maintained at all connection and splicing points.

4.2.2.4.3 Characteristic impedance

All bus sections shall present a differential characteristic impedance of $Z_w = 120,0 \Omega (\pm 10\%)$ measured with a sinusoidal signal at a frequency between 0,5 BR and 2,0 BR.

4.2.2.4.4 Cable attenuation

The cable shall attenuate a sinusoidal signal by less than 10,0 dB/km at 1,0 BR, and by less than 14,0 dB/km at 2,0 BR.

4.2.2.4.5 Distributed capacitance

The differential (wire-to-wire) distributed capacitance of the cable shall not exceed 65 pF/m at 1,0 BR.

4.2.2.4.6 Capacitive unbalance to shield

The capacitive unbalance to shield shall not exceed 1,5 pF/m at 1,0 BR.

4.2.2.4.7 Crosstalk rejection

Where two pairs of wires are carried in the same extension cable, the signal rejection from one pair to the other shall be greater than 55,0 dB in the range from 0,5 BR to 2,0 BR.

4.2.2.4.8 Shield quality

The transfer impedance of the cable, shall, at 20,0 MHz, be less than 20,0 mΩ/m, measured as specified in IEC 60096-1.

The differential transfer impedance of the cable shall be less than 2,0 mΩ/m, measured as specified in IEC 60096-1.

NOTE IEC 60096-1 mainly addresses coax cables, but it may be used for twisted pairs considering the short-circuited line pair as the inner conductor and the shield as the outer conductor.

4.2.2.4.9 Connector quality

NOTE These requirements do not apply to connectors between vehicles.

All cable connections shall provide continuity of wires and shielding, with a resistance smaller than 10,0 mΩ.

The transfer impedance of the connector, measured at 20,0 MHz, shall be less than 20,0 mΩ between one pin and shield, respectively 2,0 mΩ between two pins, measured as prescribed in IEC 60096-1.

4.2.2.5 Shielding concept

To satisfy different applications, two shielding concepts are specified:

- a grounded shield concept (preferred method) and
- a floating shield concept.

4.2.2.5.1 Grounded shield concept

When applying the grounded shield concept:

- the shields shall be connected directly to the node ground at each node;
- the jumper cables shall not establish shield continuity between vehicles, as shown in Figure 117.

NOTE The shields should be connected to ground whenever possible e.g. at vehicle ends, cabinet border etc. to loop back induced currents on short paths and to prevent them to produce EMC disturbances through the shield. This requires good conductivity of vehicle body to prevent large stray currents of traction equipment.

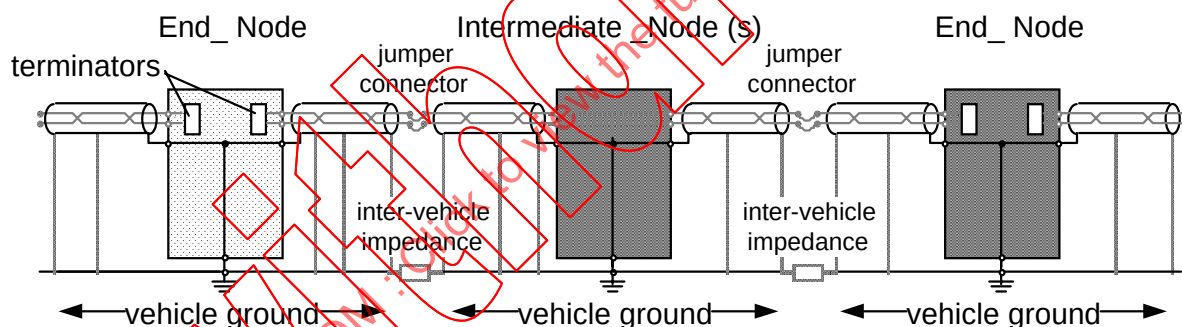


Figure 117 - Grounded shield concept

NOTE The grounded shield concept is specified by the UIC 558 leaflet.

4.2.2.5.2 Floating shield concept

When applying the floating shield concept:

- The shields shall be isolated from the ground when not connected to a node;
- The shield shall be connected on each node to the node ground by an RC circuit, consisting of a resistor of value $R_s = 47,0 \text{ k}\Omega \pm 5\%$ in parallel with a capacitor $C_s = 100,0 \text{ nF} \pm 10\%$, 750,0 V, as shown in Figure 118:

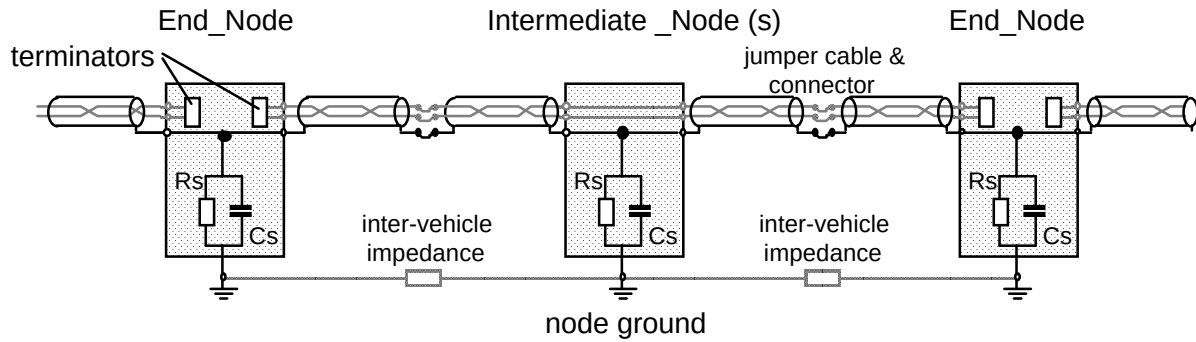


Figure 118 - Floating shield concept

4.2.2.6 Terminator

The End Nodes shall electrically terminate the two bus segments to which they are connected with a terminator.

The terminator shall be a non-polarised resistor with a value of $Z_w \pm 5\%$, and with a phase angle of less than 0,087 radians over the frequency range of 0,5 BR to 2,0 BR.

The terminator shall be isolated from the cable shield.

The terminator shall present a resistance of 2,4 k Ω to a d.c. current applied between X and Y capable of dissipating at least a sustained 1,0 W of power (even in applications which do not require fritting).

EXAMPLE - A recommended circuit is shown in Figure 119.

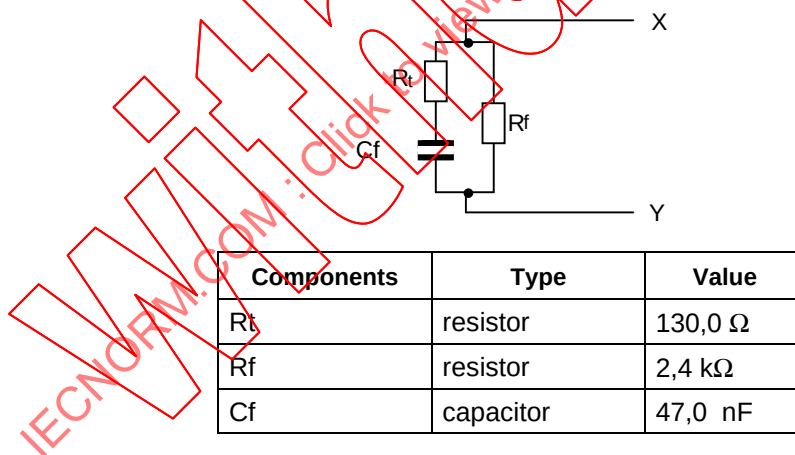


Figure 119 - Terminator

4.2.3 Medium attachment

The following Subclauses specify two methods for attaching a node:

- directly attached nodes are inserted directly in the trunk cable without a connector.
- indirectly attached nodes are attached to the trunk cable through connectors.

4.2.3.1 Node connection points identification

A node shall identify the two bus sections attached to it as 'Direction_1' and 'Direction_2', respective to that node only.

A node shall identify the two lines attached to it as 'Line_A' and 'Line_B'. If only one line is used, it shall be Line_A.

The cable attachment points to the Line Unit shall be named:

- a) A1X, A1Y and A1S for Line_A1 (Line_A in Direction_1) and
- b) A2X, A2Y and A2S for Line_A2 (Line_A in Direction_2), respectively
- c) B1X, B1Y and B1S for Line_B1 (Line_B in Direction_1) and
- d) B2X, B2Y and B2S for Line_B2 (Line_B in Direction_2).

4.2.3.2 Direct node attachment

A directly attached node shall be inserted in the cable and be attached by screws, or other fastenings which meet the electrical and mechanical requirements, as shown in Figure 120:

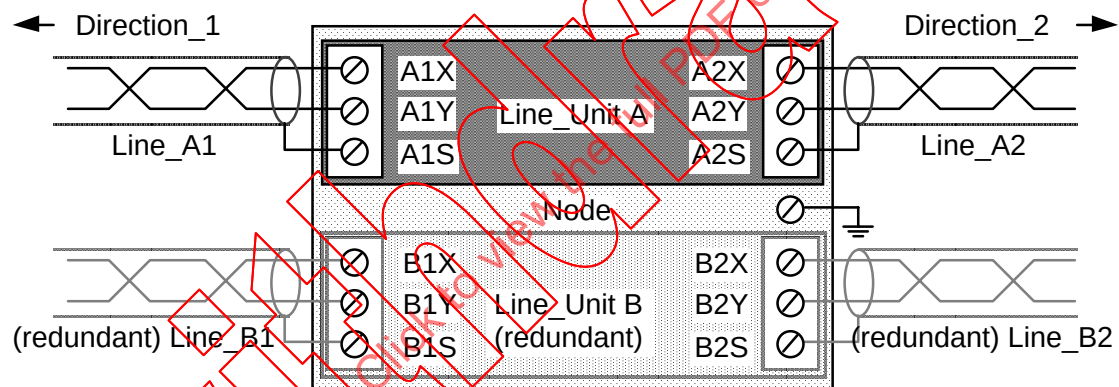


Figure 120 - Direct node attachment (optional double-line)

It shall be possible to remove the node and connect the cables of the two directions together, so as to provide cable and shield continuity.

4.2.3.3 Indirect node attachment

Indirectly attached nodes shall use two connectors in a single-line attachment, or four in a double-line attachment, as shown in Figure 121.

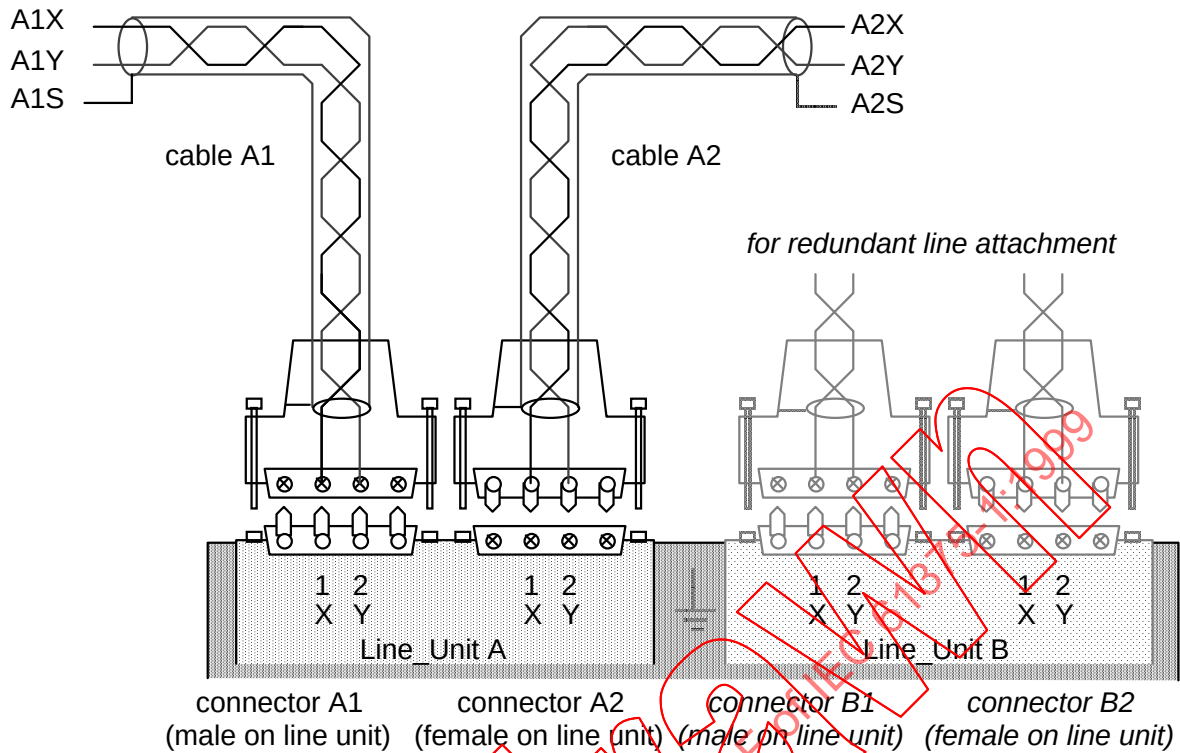


Figure 121 - Indirect attachment

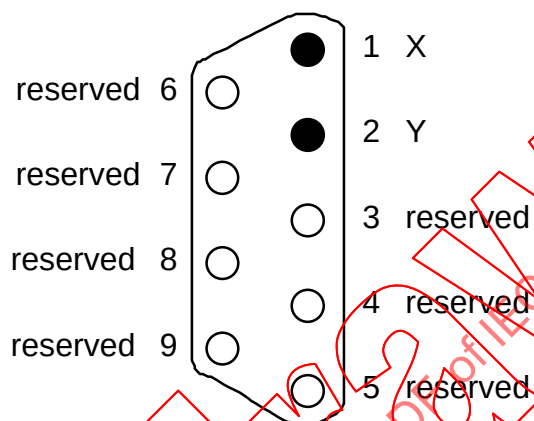
4.2.3.4 Connector (optional)

Where interchangeability is required, indirectly attached nodes shall be attached to the cable as follows:

- a) the connector shall be a subminiature-D connector (IEC 60807);
- b) the connector shall have a shielded, conductive casing, such that:
 - for the grounded shield concept, this casing shall be connected to the cable shield and make an electrical contact with the receptacle when fastened.
 - for the floating shield concept, the casing may be isolated from the cable shield.
- c) the connector shall have metric screws;
- d) the connector shall have the following polarity and arrangement:
 - Direction_1 shall use the male connector on the Line Unit and the female on the cable;
 - Direction_2 shall use the female connector on the Line Unit and the male on the cable;
 - When connectors of the same line are arranged vertically, Direction_1 shall be the upper connector, Direction_2 shall be the lower connector, Line_A shall be the upper pair;
 - When connectors of the same line are arranged horizontally, Direction_1 shall be the left connector, Direction_2 shall be the right connector, looking towards the node, Line_A shall be the upper pair.
- e) it shall be possible to connect and fasten the cable connectors of the two directions together, so as to provide cable and shield continuity;
- f) the connector (male or female) shall have the pin assignment specified in Table 60, as shown in Figure 122.

Table 60 - WTB connector pin assignment

1	X positive wire	6	reserved
2	Y negative wire	7	reserved
3	reserved for shield	8	reserved
4	reserved	9	reserved
5	reserved		

**Figure 122 - WTB connector, front view**

NOTE It is preferable for the floating shield concept to electrically separate the receptacle from the node case.

4.2.4 Node specifications

4.2.4.1 Node elements

A node shall be attached to the medium by its Medium Attachment Unit (MAU).

The MAU for single-line attachment shall comprise:

- a) a Line Unit;
- b) a Direction Commutator;
- c) a Main Channel and an Auxiliary Channel.

EXAMPLE - A MAU with switches in Intermediate Setting is shown in Figure 123.

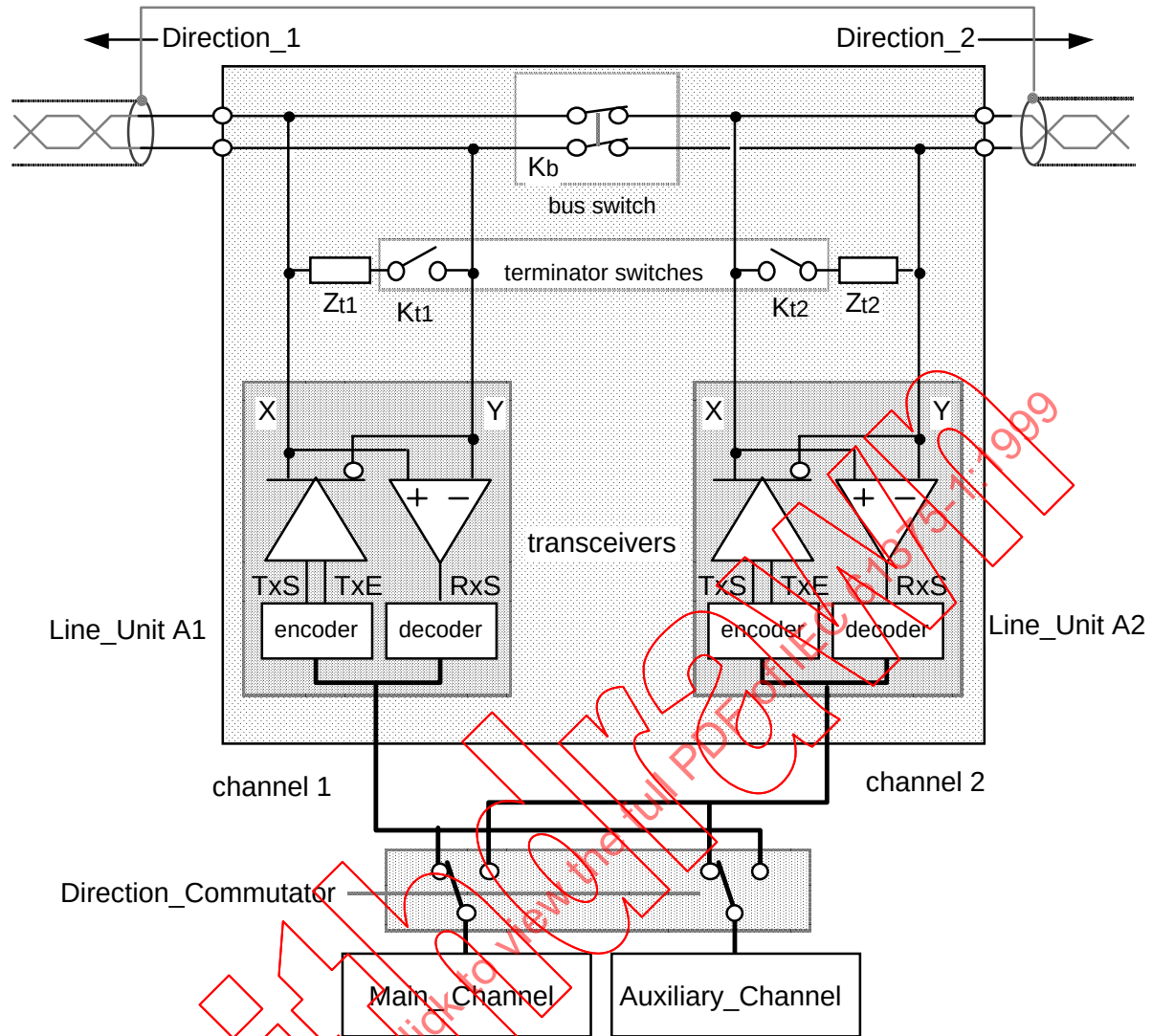


Figure 123 - Example of MAU Structure

4.2.4.1.1 Line Unit elements

A Line Unit shall consist of:

- a bus switch (K_b), which connects or disconnects the two directions;
- a terminator switch for each direction (K_{t1} , K_{t2}), which inserts a terminator (Z_{t1} , Z_{t2}) in a node in End Setting or removes it in a node in Intermediate Setting;
- two transceiver (transmitter/receiver) circuits, one for each direction. Each transmitter is controlled by the binary signals TxS (signal) and TxE (enable). The output of the receiver is RxS (a digital or analogue signal). The transceivers are isolated galvanically from the line by suitable means, for instance by a transformer;
- two Manchester encoder/decoders, one for each transceiver, which may be integrated in their respective transmitter or receiver. The output and input of the Manchester encoder/decoder are specified as a modem interface;
- protection circuits against over-voltage/short circuit, connected to the isolation switches (not shown in Figure 123).

NOTE 1 A node attached to a redundant medium has two Line Units.

NOTE 2 Switches K_b and K_{t1} or K_{t2} may be contacts of the same mechanical relay if such relays are used.

4.2.4.1.2 Main Channel and Auxiliary Channel

The Main Channel and the Auxiliary Channel shall both be capable of sending and receiving HDLC Frames and control signals to and from the Line Units.

NOTE As the Auxiliary Channel only receives and sends frames for detecting additional nodes and for receiving its address, its operation may be simplified with respect to the Main Channel.

4.2.4.1.3 Direction Commutator

The Direction Commutator shall connect the Main Channel to Direction_1 and the Auxiliary Channel to Direction_2, or vice versa.

NOTE The Direction Commutator is not necessarily a physical element, it may be implemented in logic or software.

4.2.4.2 Node and switch settings

This subclause describes the node characteristics in the two settings: Intermediate Setting or End Setting.

4.2.4.2.1 Intermediate Setting

A node shall, in the Intermediate Setting :

- a) establish continuity between the two line segments (Kb closed);
- b) remove both Terminators Zt1 and Zt2 (Kt1 and Kt2 are open);
- c) connect the Main Channel to the line over either transceiver 1 or transceiver 2;
- d) shut down the Auxiliary Channel and the unused transceiver.

NOTE The Intermediate Setting is taken by a non-operational (disabled or powered-off) node and by nodes not situated at the end of the bus.

4.2.4.2.2 End Setting

A node shall, in the End Setting :

- a) isolate both segments (Kb is open);
- b) insert both Terminators (Kt1 and Kt2 are closed);
- c) connect the Auxiliary Channel towards one direction and the Main Channel towards the other direction.

NOTE The End Setting is taken by an unnamed node, a node at the end of the bus (in particular, a master without a slave) or a node in the sleep mode.

4.2.4.3 Duplicated Line Units (option)

In case this option is used, the following specifications apply:

- MAUs designed for duplicated attachment shall be attached to Line_A and Line_B over separate Line Units;
- the node settings (End Setting or Intermediate Setting) shall apply to both Line Units equally;
- it shall be possible to remove one line while keeping the other line operational.

EXAMPLE - A Medium Attachment Unit for redundant line operation is shown in Figure 124. The switchover logic allows signals to be received either from Line_A or Line_B.

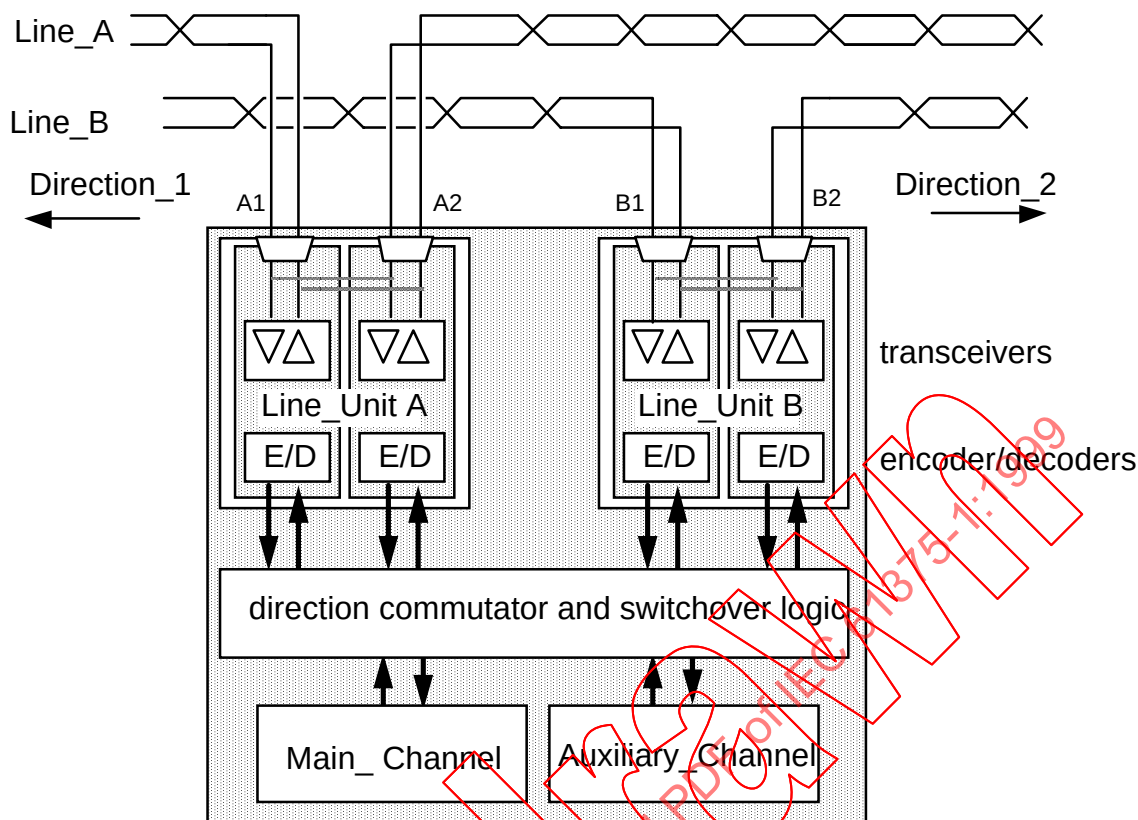


Figure 124 - Node with redundant Line Units

4.2.5 Line Unit specifications

Although only Line Unit A is mentioned, these specifications also apply to Line Unit B if a duplicated medium is used.

4.2.5.1 Galvanic separation

The isolation voltage and isolation resistance, between node casing and any of the points: A1X, A1Y, A2X or A2Y, shall exceed the value specified in IEC 60571.

NOTE These values are in the current edition 0,50 kVrms and 1,0 MΩ.

4.2.5.2 Insertion losses of a Line Unit

4.2.5.2.1 Attenuation measurement

For insertion loss measurement, the sinusoidal signal of a generator (of internal impedance = Z_t) is applied through 20,0 m of cable to the points A1X and A1Y and measured by a voltmeter (connected in parallel with an impedance Z_t) at the extremity of other 20,0 m of cable attached to the points A2X and A2Y, or vice-versa, as shown in Figure 125.

The attenuation is defined as the ratio expressed in dB of two differential voltages:

- a first voltage being set to 4,0 V_{pp} when the node is removed and the cable connector coupled;
- a second voltage being measured when the node is inserted (in End Setting or Intermediate Setting, according to the test).

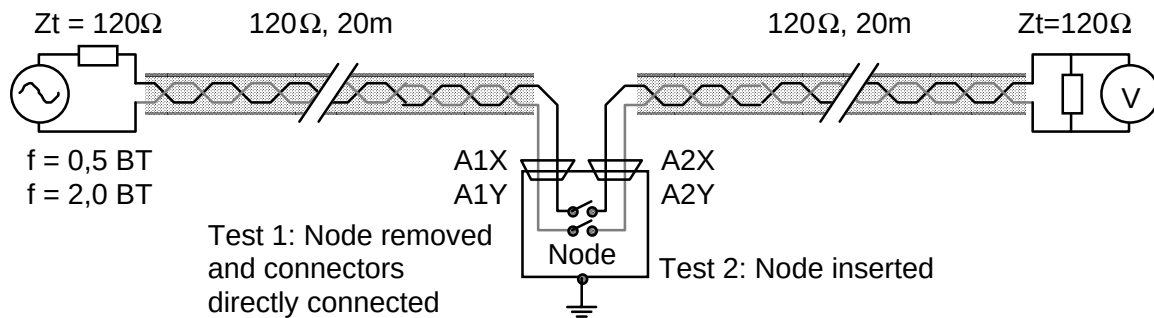


Figure 125 - Attenuation measurement

4.2.5.2.2 Node In End Setting

A Line Unit in the End Setting (Kb open, Kt1 and Kt2 closed) shall present to a 1,0 BR signal applied between A1X and A1Y, or between A2X and A2Y, the impedance corresponding to the terminator specified in 4.2.2.6.

A Line Unit in the End Setting shall attenuate by more than 55,0 dB a signal applied between A1X and A1Y and measured between A2X and A2Y or vice-versa.

4.2.5.2.3 Node In Intermediate Setting

A Line Unit in Intermediate Setting (Kb closed, Kt1 and Kt2 open), either:

- with receiver in normal operation and with transmitter in the high-impedance state, or
- with no power applied to either receiver or transmitter,

shall attenuate a sinusoidal signal:

- by less than 0,3 dB between 0,5 BR and 1,0 BR; and
- by less than 0,4 dB up to 2,0 BR.

The node shall present a resistance of at least 1 MΩ against a positive or a negative DC voltage of 48,0 V applied between A1X and A1Y or applied between A2X and A2Y.

4.2.5.3 Switches specifications

All switches connected to the bus line (Kb, Kt, etc.):

- shall present an isolation of at least 500,0 Vrms in the open setting;
- shall present an initial contact resistance of less than 0,050 Ω in the closed setting;
- shall be specified for a contact resistance of less than 0,100 Ω in the closed setting after 10^7 cycles;
- shall switch from one setting to the other in less than 10,0 ms, including bounce time.

NOTE The relay may be of the solid-state or mechanical type.

4.2.5.4 Shield connection to a Line Unit

At each node, the shields of the cables in Direction_1 and Direction_2 shall be connected together through the receptacles with a contact resistance of less than 0,010 Ω.

A Line Unit shall provide means to connect the shields to the node case both:

- a) directly through a low impedance, as specified in 4.2.2.5.1 (grounded shield);
- b) through the RC network specified by 4.2.2.5.2 (floating shield).

EXAMPLE - The principle of shield attachment in a node is shown in Figure 126.

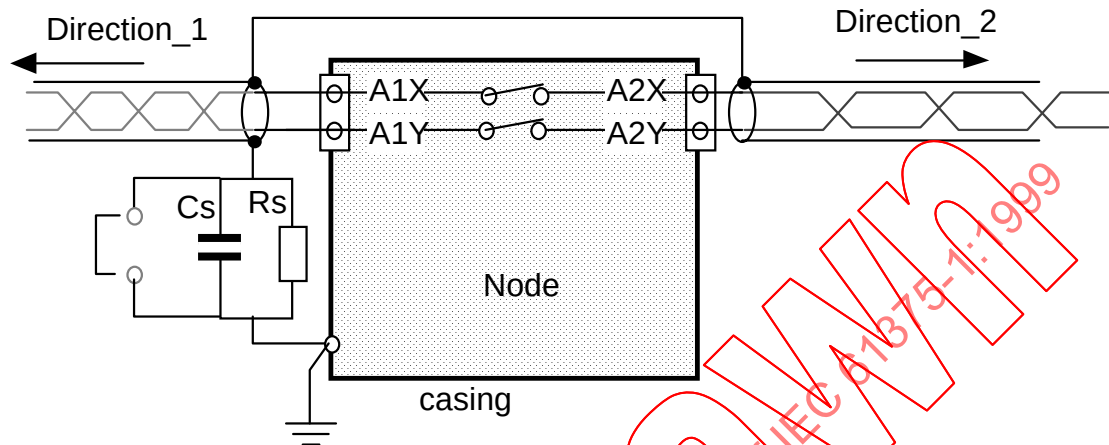


Figure 126 - Shield grounding in the Line Unit

4.2.5.5 Fritting (option)

To overcome contact oxidation in relays and connectors, a node may use fritting, consisting in applying a continuous voltage between the wires X and Y of either direction.

If the fritting option is used, the specifications of this subclause apply.

NOTE 1 The use of fritting is not specified for a non-redundant medium.

NOTE 2 Nodes not using fritting and nodes using it may be mixed on the same bus.

4.2.5.5.1 Fritting source and load

A node which supports fritting shall provide a fritting voltage source and a fritting voltage load for each direction.

In case of a redundant physical medium, a node shall provide two independent fritting voltage sources, one for each direction, and shall provide the loads for the fritting voltage of another node as shown in Figure 127.

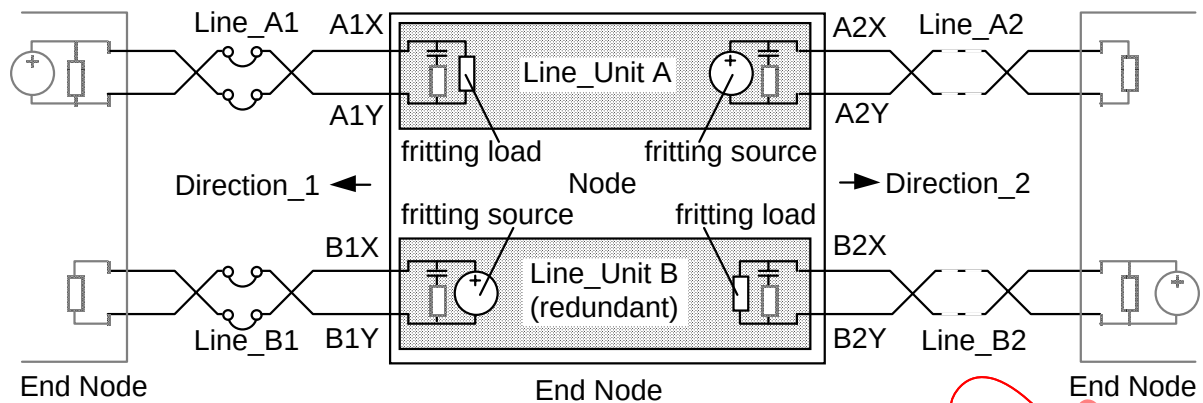


Figure 127 - Fritting source and load

The positive pole of the fritting source shall be connected to A2X, respectively B1X.

The negative pole of the fritting source shall be connected to A2Y, respectively B1Y.

The fritting source shall supply a direct voltage of 48,0 V $\pm 20\%$ measured at the connection points (A2X / A2Y respectively B1X / B1Y), when the corresponding line, with its terminator inserted, is connected to the terminator specified in 4.2.2.6 or left open.

The ripple of the fritting source shall be below 0,100 Vpp in the range of 0,5 BR to 2,0 BR.

The current delivered by the frittig source shall not exceed 80,0 mA, DC.

The fritting source shall have an input to output isolation allowing it to conform to IEC 60571.

The fritting source shall be decoupled from the line, for instance by an inductor of 0,10 H or any other arrangement which meets the insertion losses of a node.

The switch-on time constant of the source shall be in the range of 0,5 ms to 5,0 ms.

The switch-off time constant of the source shall be in the range of 0,5 ms to 5,0 ms.

The attenuation between the two fritting voltage sources in the same node shall be greater than 50,0 dB in the range between 0,5 BR and 2,0 BR.

4.2.5.5.2 Applying fritting

An End Node shall switch on the fritting source over its active Auxiliary Channel (an unnamed node has both directions as active Auxiliary Channels, a node in sleep mode has no active Auxiliary Channel).

NOTE 1 A node may pulse its fritting source, for instance to reduce consumption.

NOTE 2 The switching of the fritting source over the Main Channel is allowed as long as it complies with the electromagnetic interference levels.

4.2.6 Transceiver specifications

A MAU attached to a redundant bus has four transceivers, named A1, A2, B1 and B2. In a non-redundant arrangement, only transceivers A1 and A2 are used. The following specifications apply to any of them.

4.2.6.1 Conventions

Unless otherwise specified, the following default measuring conditions hold:

- a) The characteristics of a transceiver are measured at the X and Y points where the cable sections are attached to the node;
- b) All voltages are measured as differential voltage between X and Y, ($U_x - U_y$);
- c) When measuring a transmitter, the circuit of the receiver is in the normal receiving state. When measuring a receiver, the circuit of the transmitter is in a high impedance state;
- d) All resistor values are $\pm 1\%$, all capacitor values are $\pm 10\%$.

4.2.6.2 Transmitter

4.2.6.2.1 Test circuit for transmitter

To approximate the loading of the transmitter with a cable and nodes, four test circuits are specified:

- a) The light test circuit simulates an open line (as in a node in End Setting). The value of the total resistive load is equal to that of the terminator;
- b) The heavy test circuit simulates a fully loaded bus. The value of the total resistive load is equal to 0,42 of that of the terminator;
- c) The idling test circuit simulates a cable of 860,0 m without resistive loads. The capacitors have a value of $1,3 \text{ nF} \pm 10\%$ each, the resistors a value of $27,0 \Omega \pm 1\%$ each;
- d) The short test circuit simulates a line failure. It consists only of a current measurement circuit.

These circuits are shown in Figure 128.

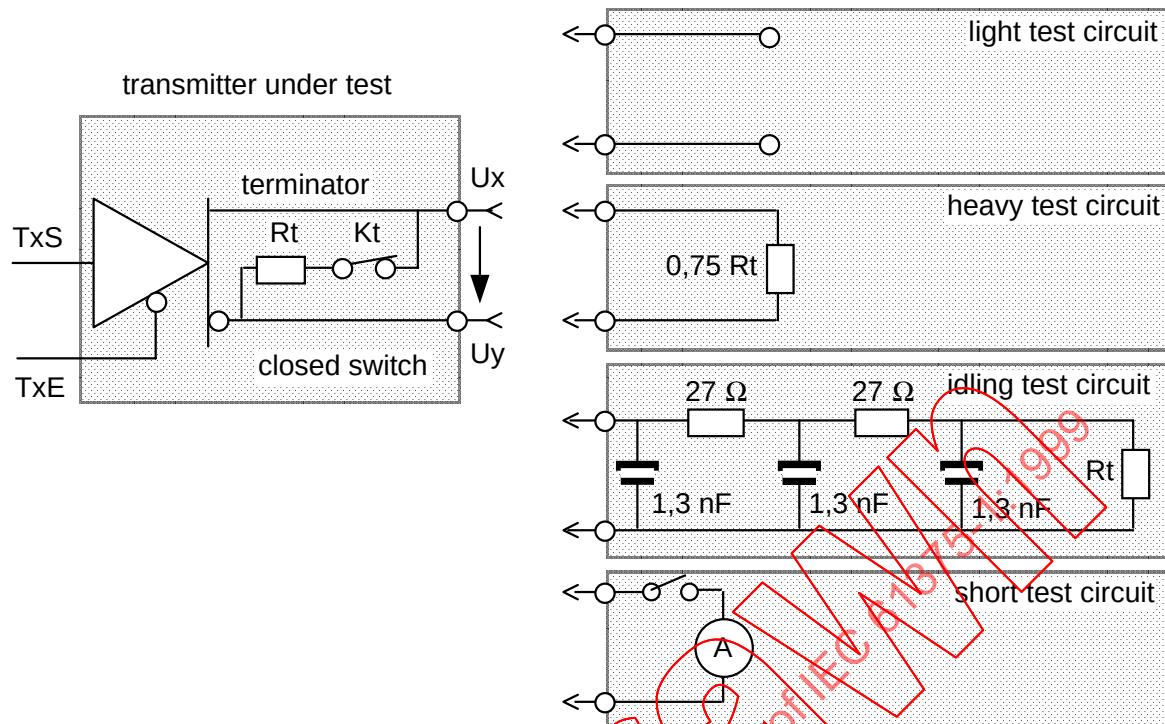


Figure 128 - Transmitter test circuits

The measurement is made with the node in End Setting (K_b open, K_t closed).

NOTE The terminator of the Line Unit is considered in the test circuit specification.

4.2.6.2.2 Transmitter output Signal

NOTE Due to the data encoding used, the transmitter generates, between Preamble and End Delimiter, pulses which are either one bit in length (1,0 BT) or a half-bit in length (0,5 BT).

These specifications apply to both 0,5 BT and 1,0 BT pulses, positive or negative.

The transmitter shall be a differential driver.

The output signal is the differential voltage ($U_x - U_y$) at the connection point of a node.

When connected to either the heavy and the light test circuit defined in 4.2.6.2.1, the transmitter shall comply with the following specifications, as shown in Figure 129:

- The output signal shall be alternatively positive and negative;
- The amplitude of the output signal shall be at least $\pm 3,0$ V with the heavy test circuit and at most $\pm 7,0$ V with the light test circuit;
- The peak amplitude is defined as the maximum amplitude of the output signal. The signal shall not drop by more than 20% from this peak amplitude until $0,100 \mu\text{s}$ from the next expected zero-transition; The ringing of the amplitude during this time, relative to the average voltage drop, shall not exceed 5% of the peak value;
- The slew rate of the output signal shall be less than $0,20$ V/ns at any time and more than $0,03$ V/ns within $100,0$ ns of the zero-crossing;

- e) The overshoot of the output signal, defined as the ratio of the maximum amplitude to the stationary amplitude shall not exceed 10 % of its stationary amplitude;
- f) The edge distortion of the output signal, defined as the time difference between the idealised and the actual zero crossing, shall not exceed $\pm 2\%$ of one Bit Time.

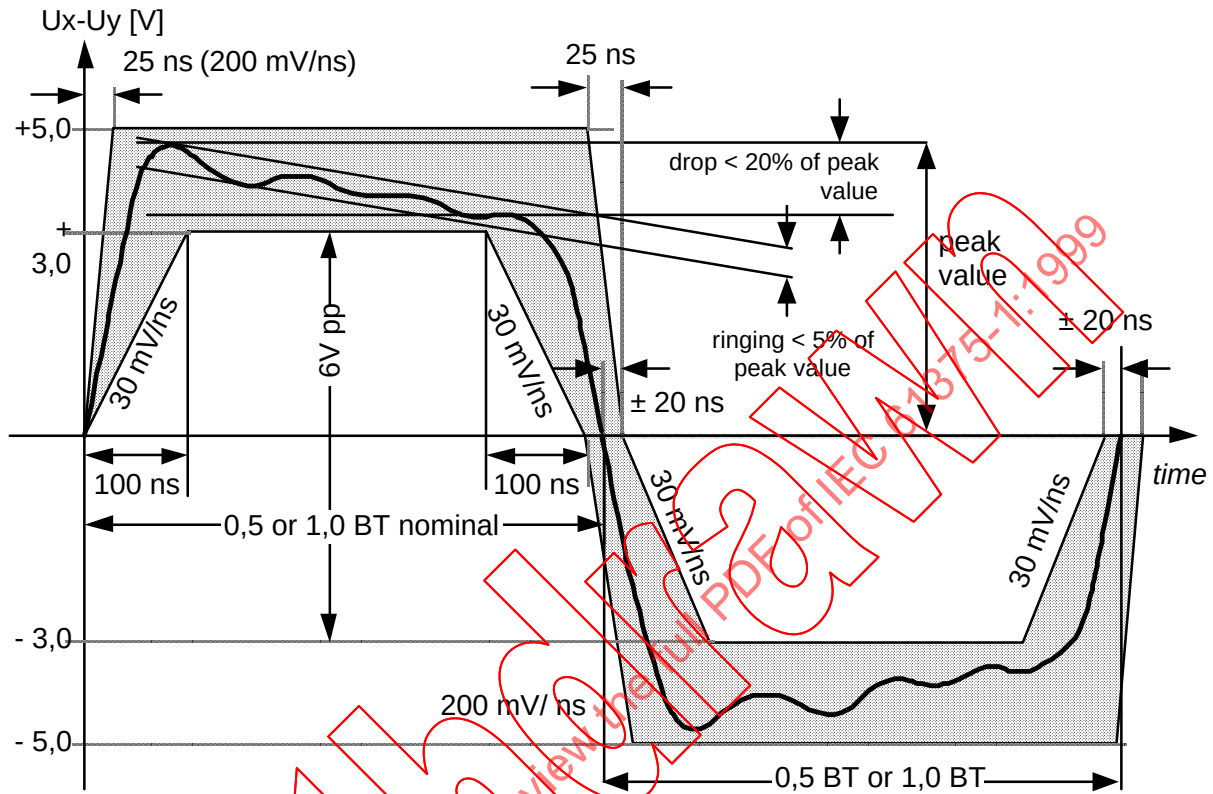


Figure 129 - Pulse wave form at transmitter

NOTE The voltage drop is expected to occur because of the fringing capacitor in series with the transformer.

4.2.6.2.3 Transmitter noise

Any noise generated by a transmitter which is not transmitting shall not exceed a value of 5,0 mVrms over the frequency range 1,0 kHz to 4,0 BR.

4.2.6.2.4 Transmitter end of frame

The end of frame produced by the transmitter shall be tested under the following conditions:

- a) the transmitter transmits the longest possible frame;
- b) the Frame_Data bits are a pseudo-random sequence of '1' and '0' symbols;
- c) the frame is closed with the End Delimiter symbol as specified in 4.3.1.4;
- d) the transmitter drives the idling test circuit of 4.2.6.2.1;
- e) the average differential amplitude is greater than 4,5 V before the transmitter is disabled,

Under these conditions, the output signal shall remain within the following limits, as shown in Figure 130:

- i. 100,0 ns after the last negative-to-positive transition and for $2,0 \text{ BT} \pm 100 \text{ ns}$, the output signal shall remain above 0,300 V;

- ii. within 3,0 BT after the last negative-to-positive transition, the output signal shall fall below 1,100 V;
- iii. within 20,0 μ s, starting when the output signal first reaches 1,100 V, the output signal amplitude shall not exceed 0,100 V;
- iv. within 64,0 μ s, starting when the output signal first reaches 1,100 V, the output signal amplitude shall not exceed 0,025 V.

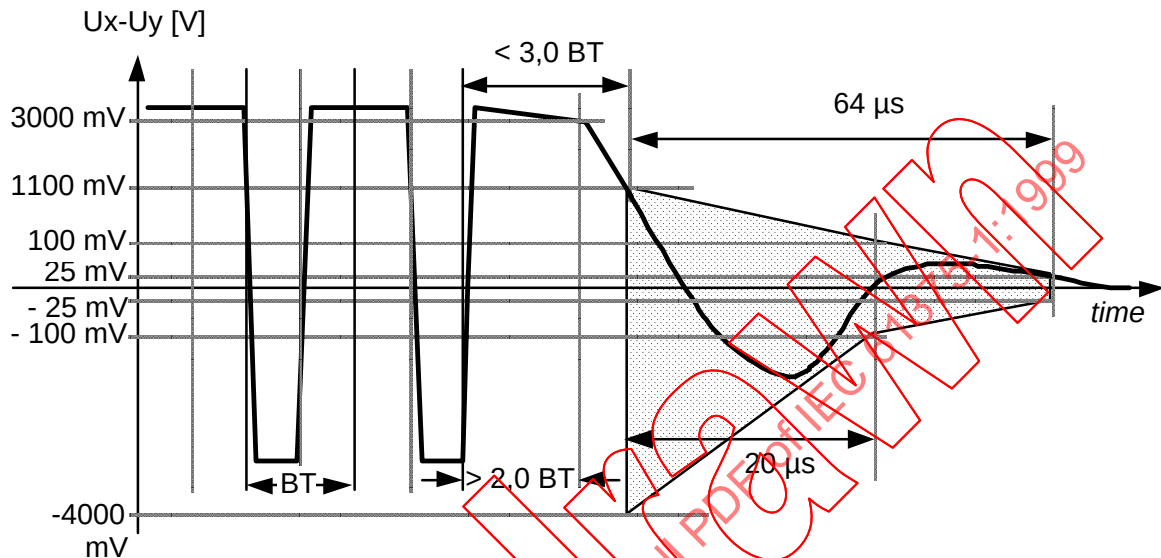


Figure 130 - Signal and idling at transmitter

NOTE Line ringing after transmitter idling can be minimised by balancing the signal in each bit cell. A further reduction can be achieved by balancing the End Delimiter as specified in 4.3.1.4.

4.2.6.2.5 Transmitter fault tolerance

A transmitter, either when enabled and not, shall tolerate the application of the short test circuit (4.2.6.2.1) at the connection point until thermal stability is reached, and shall resume normal operation after the short test circuit is removed.

The short circuit current shall not exceed 1,0 A.

NOTE For conformance testing, thermal stability is considered to be reached after 1h.

4.2.6.2.6 Transmitter anti-jabber

Each transmitter shall contain an independent circuit to decouple the transmitter from the bus line if the transmission duration exceeds the value T_{jabber} , equal to the duration of longest possible frame (including Preamble, End Delimiter and bit-stuffing) + 20%.

4.2.6.3 Receiver specifications

4.2.6.3.1 Receiver test signal (guideline)

The characteristics of a receiver are tested by applying a test signal to the connection points and observing the frames received, either through direct access to the node, or through a node-internal response, such as response to a master frame.

To test the receiver, a test signal with the following shape, as shown in Figure 131, is applied at the connection point:

- When the amplitude of the test signal is less than 0,100 V, the slope of the test signal exceeds 2,0 mV/ns;
- To test sensitivity, the test signal remains above 0,300 V for a time period that starts after 100,0 ns of the preceding zero-crossing and that lasts at least $(0,5 \text{ BT} - 350,0 \text{ ns})$, respectively $(1,0 \text{ BT} - 0,350 \mu\text{s})$, whilst its peak amplitude will vary between 0,330 V and 5,00 V;
- To test insensitivity, the test signal amplitude does not exceed a prescribed value;
- To test edge distortion, the crossing of the zero voltage is varied by a prescribed value.

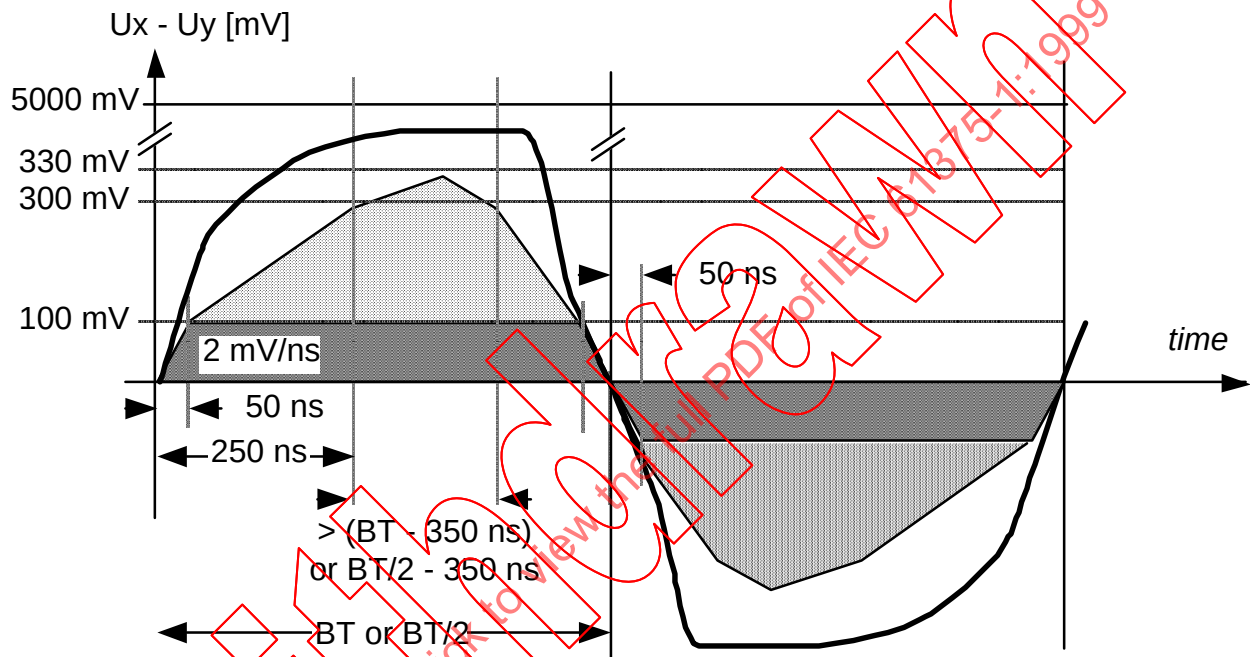


Figure 131 - Receiver signal envelope

A frame error is detected as a missing or a non valid frame (see 4.3.1.5.3), incorrect frame size or incorrect Frame_Data bits (FCS error).

4.2.6.3.2 Receiver polarity

A HIGH level on TxS shall correspond to a positive differential voltage ($U_x - U_y$), which in turn shall correspond to a HIGH level on the RxS signal of a receiver.

A LOW level on TxS shall correspond to a negative differential voltage ($U_x - U_y$), which in turn shall correspond to a LOW level on the RxS signal of a receiver.

The state of RxS is undefined when the line is idle.

4.2.6.3.3 Receiver sensitivity

A receiver receiving frames containing 64 random data bits, at the rate of 1000 frames per second shall detect no more than three frame errors in $3 \times 3 \times 10^{+6}$ frames, when the amplitude of the test signal defined in 4.2.6.3.1 varies between its minimum and maximum value.

NOTE According to Figure 131, a receiver is able to operate over a voltage range of 0,500 V to 0,330 V, which is about 23,6 dB. This leaves a noise margin of about 4,0 dB, considering that the medium attenuation is specified to be less than 20,0 dB in 4.2.2.3.3.

4.2.6.3.4 Receiver insensitivity

The receiver shall not decode a valid frame (see 4.3.1.5.3) when the test signal (4.2.6.3.1) is less than 0,100 V.

4.2.6.3.5 Receiver edge distortion

A receiver receiving frames containing 64 random data bits, at the rate of 1000 frames per second shall detect no more than three frame errors in $3 \times 3 \times 10^6$ frames when the test signal edges cross the zero voltage line at random within $\pm 10\%$ of 1,0 BT around the expected crossing, as shown in Figure 132.

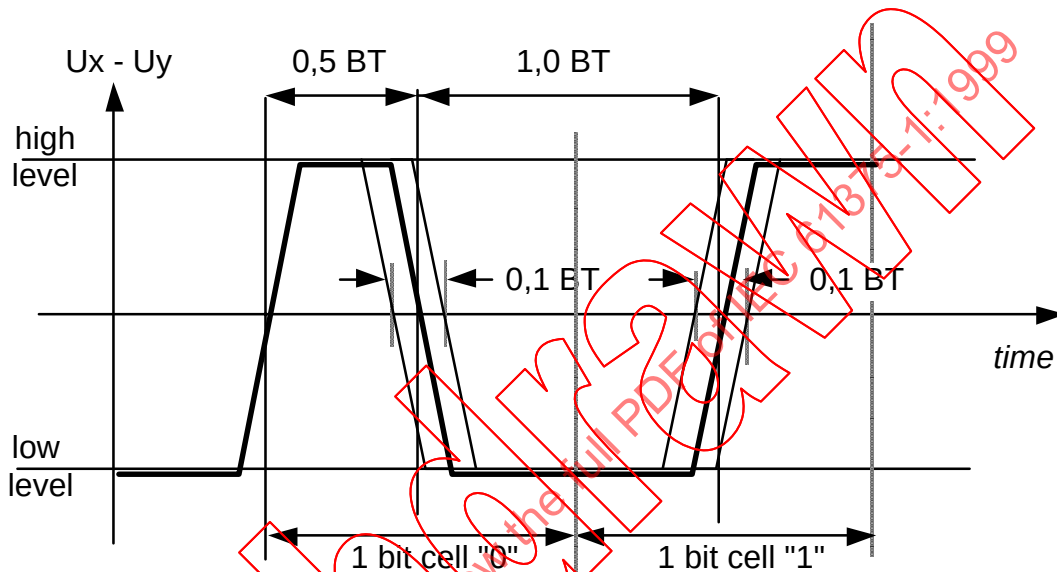


Figure 132 - Receiver edge distortion

4.2.6.3.6 Receiver noise rejection

A receiver receiving frames containing 64 random data bits, at the rate of 1000 frames per second, and with a signal amplitude of 0,700 V (1,400 V peak-to-peak) shall detect no more than 3 frame errors 3×10^6 frames when operating:

- in the presence of a common-mode sinusoidal signal applied between casing and both data wires with an amplitude of 4,000 Vrms in the frequency range of 65,0 Hz to 1,5 MHz; or
- in the presence of an additive quasi-white Gaussian noise (applied between X and Y) distributed over a bandwidth of 1,0 kHz to 4,0 MHz at an amplitude of 0,140 V rms.

4.3 Medium-dependent signalling

4.3.1 Frame encoding and decoding

4.3.1.1 Conventions

Encoding and decoding assume that the sent or received signal is binary, with no biased level. The received level is undefined when the line is idle.

RxS represents the idealised (analogue) received signal from the line.

A frame shall be transmitted as a sequence of positive and negative levels beginning with a Preamble and ending with an End Delimiter, before it returns to the idle state, as shown in Figure 133.

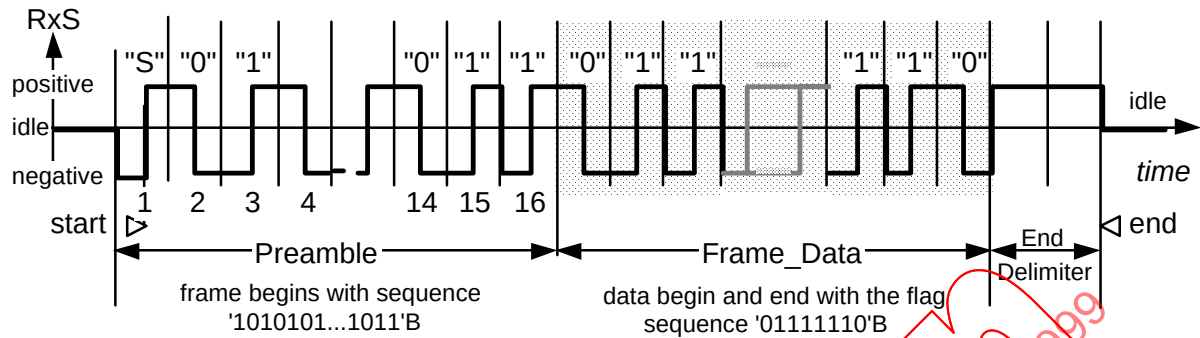


Figure 133 - Idealised frame on the line (16 bit Preamble shown)

4.3.1.2 Bit encoding

Preamble and Frame_Data bits shall be encoded by a Manchester code, as shown in Figure 134:

- A '1' bit shall be encoded by a negative level during the first half of a bit cell going to the positive level in the middle of the cell;
- A '0' bit shall be encoded by a positive level during the first half of a bit cell going to the negative level in the middle of the cell.

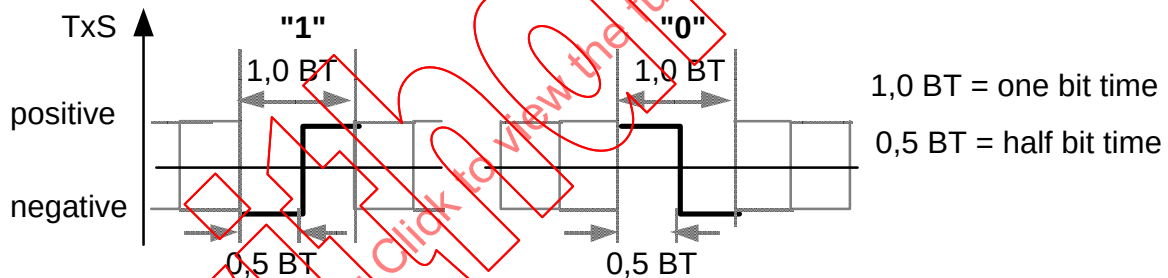


Figure 134 - Bit encoding

Edge distance shall be either a full bit time ('0' follows '1' or '1' follows '0') or a half Bit Time (sequence of '0' or sequence of '1') until the End Delimiter of the frame is reached.

4.3.1.3 Preamble encoding

A frame shall be headed by a Preamble, consisting of a sequence of bits beginning with a start bit S sent as a '1' bit, followed by pairs of ('0', '1') bits and closed by a '1' bit as shown in Figure 135.

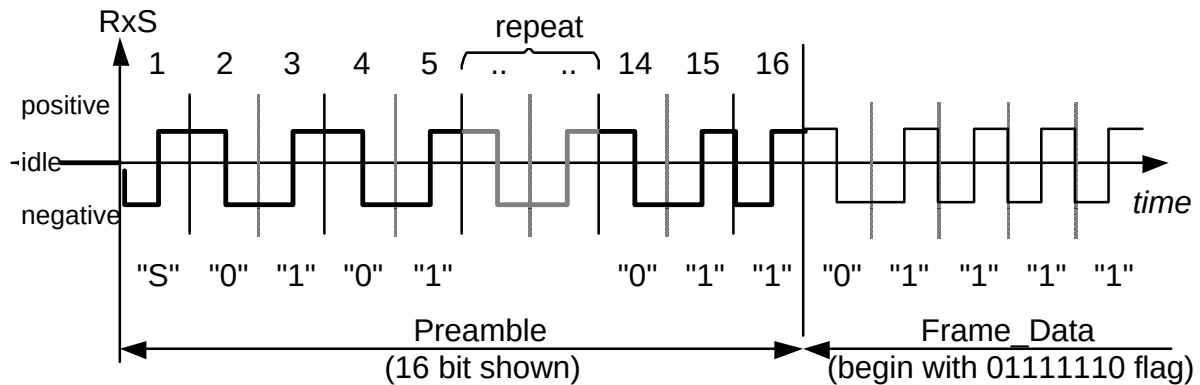


Figure 135 - Preamble

There shall be a minimum of seven and a maximum of 15 pairs of ('0','1') between the start bit and the closing '1'.

The decoder may check the polarity of RxS by decoding the Preamble, but it shall not automatically invert the signal if X and Y have been accidentally interchanged.

NOTE In the following, only a 16-bit Preamble will be considered.

4.3.1.4 End Delimiter

The frame shall be closed by an End Delimiter which maintains the line at positive level for the duration of 2,0 BT.

A negative level of 2,0 BT with a duration of 2,0 BT may be appended after the positive level to compensate for the unbalance, as shown in Figure 136.

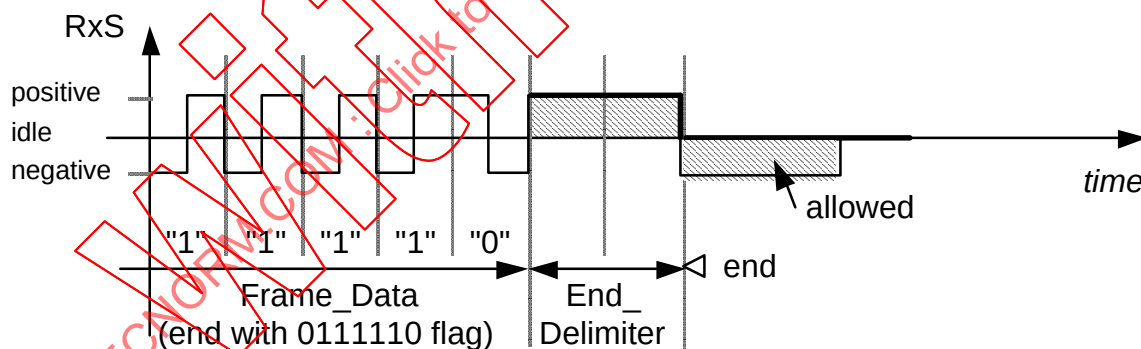


Figure 136 - End Delimiter

NOTE 1 An End Delimiter without a negative level causes an unbalance which lets the line ring (see 4.2.6.2.4). The compensation pulse is highly recommended, but not mandatory, to allow the use of commercial circuits which do not generate it.

NOTE 2 Due to the HDLC flag (see 4.4.1), the last bit of a frame is a '0'.

4.3.1.5 Signal quality supervision

The following specifications assumes that the decoder generates two signals, called Carrier_Sense (CS) and Signal_Quality_Error (SQE), for signal quality supervision and redundancy switchover.

4.3.1.5.1 Carrier_Sense

The decoder shall assert CS within 0,5 BT after it detects the last received bit of a Preamble according to 4.3.1.3.

The decoder shall negate CS within 0,5 BT after it detects an End Delimiter or detects bits which are neither a '0', nor a '1', nor an End Delimiter.

4.3.1.5.2 Signal_Quality_Error

The decoder shall negate SQE within 0,5 BT after it detects the last received bit of a Preamble according to 4.3.1.3.

The decoder shall assert SQE within 0,5 BT if it detects bits which are neither a '0', nor a '1', nor an End Delimiter while CS is asserted.

4.3.1.5.3 Valid frame

A frame shall be defined as valid if it consists of a Preamble, a number of '0' and '1' bits and an End Delimiter.

EXAMPLE - A valid frame and the corresponding CS and SQE signals is shown in Figure 137.

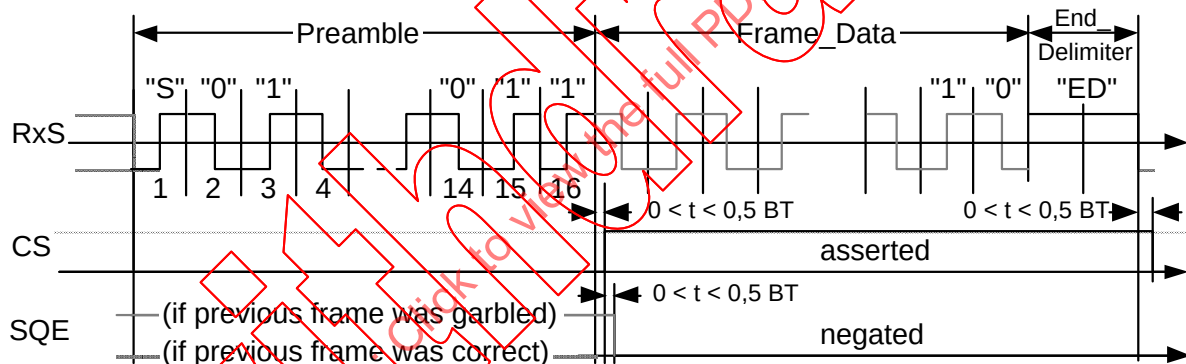


Figure 137 - Valid frame, RxS, CS and SQE signals

For the purpose of redundancy control, a valid frame shall consist of a Preamble followed by a sequence of at least eight data bits.

4.3.1.5.4 Not valid frame

A frame shall be defined as not valid if SQE is asserted for a time longer than 0,5 BT while CS is asserted.

EXAMPLE - The timing of the signals when a garbled frame is received is shown in Figure 138.

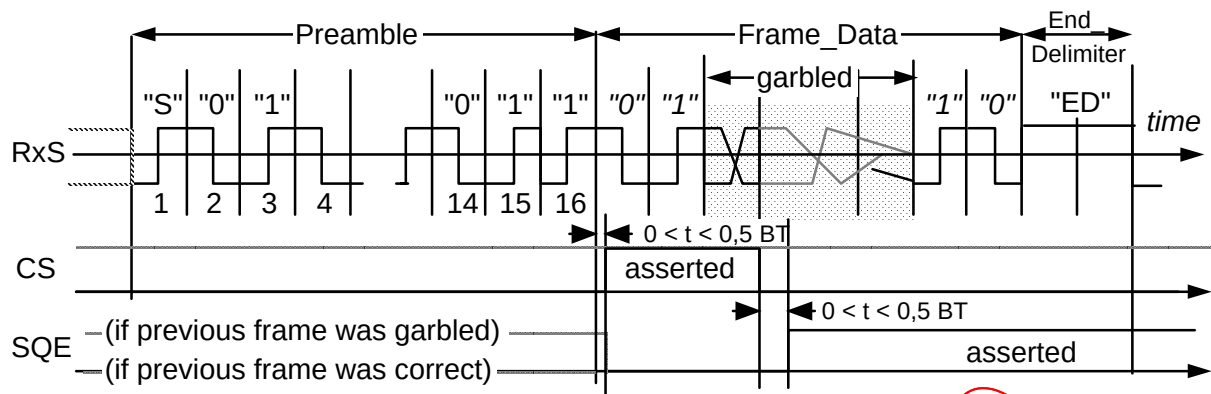


Figure 138 - Garbled frame, RxS, CS, SQE signals

If SQE becomes active, the decoder shall ignore all data until the next Preamble is received.

4.3.2 Duplicated line handling (option)

This specification defines an optional redundancy scheme. In case this option is used, the following specifications apply to both Direction_1 and Direction_2 and to both Line_A and Line_B.

4.3.2.1 Principle

A node transmits the same data simultaneously over Line_A and Line_B and a node accepts data from one line, called the Trusted Line, while it monitors the other line, called the Observed Line.

Each node selects its Trusted Line and Observed Line based on the signals generated by its own physical layer, or upon request of its link layer, independently from other nodes.

To remain independent of the medium, the selection of the Trusted Line relies on signals which the Line Unit generates, as they are defined in the Line Unit interface.

4.3.2.2 Skew

Since the signals on the Line_A and Line_B are subject to different delays, their skew (timing difference) differs at the transmitter, at the receiver or anywhere along the line, as shown in Figure 139.

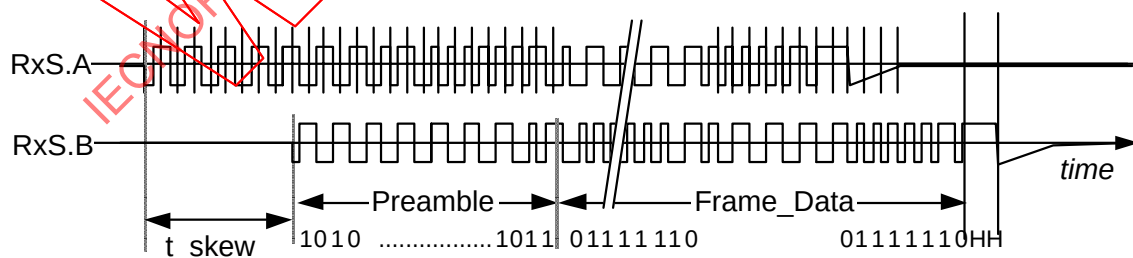


Figure 139 - Redundant Lines (as seen at a receiver)

4.3.2.3 Redundant transmission

A MAU with redundant Line Units shall transmit the same signal over Line_A and Line_B (Line_A1 and Line_B1, or Line_A2 and Line_B2).

The timing difference between the signal measured at the output of the Line Units between Line_A and Line_B in the same direction, shall not exceed $T_{skew\_t} = 1,0 \mu s$.

4.3.2.4 Redundant reception

4.3.2.4.1 Skew at reception

A receiver shall permit a maximum skew at the receiver, T_{skew_r} , of $32,0 \mu s$.

4.3.2.4.2 Line_Disturbance

There shall be a 'Line_Disturbance' signal for each of the bus sections attached to a node, called DA1 and DA2 for Line_A, and DB1 and DB2 for Line_B.

The 'Line_Disturbance' signal of a line shall be asserted if:

- the decoder asserts 'Signal_Quality_Error' on that line; or
- the decoder does not generate a 'Carrier_Sense' within T_{skew_r} after the Line Unit of the redundant line asserted its 'Carrier_Sense' (missing frame).

The 'Line_Disturbance' signal shall be negated:

- when the decoder receives a valid frame as defined in 4.3.1.5.3.

In a non-redundant mode, the unused line shall be considered as permanently disturbed.

EXAMPLE - A case in which all frames on Line_A1 are valid, but Line_B1 is disturbed is shown in Figure 140.

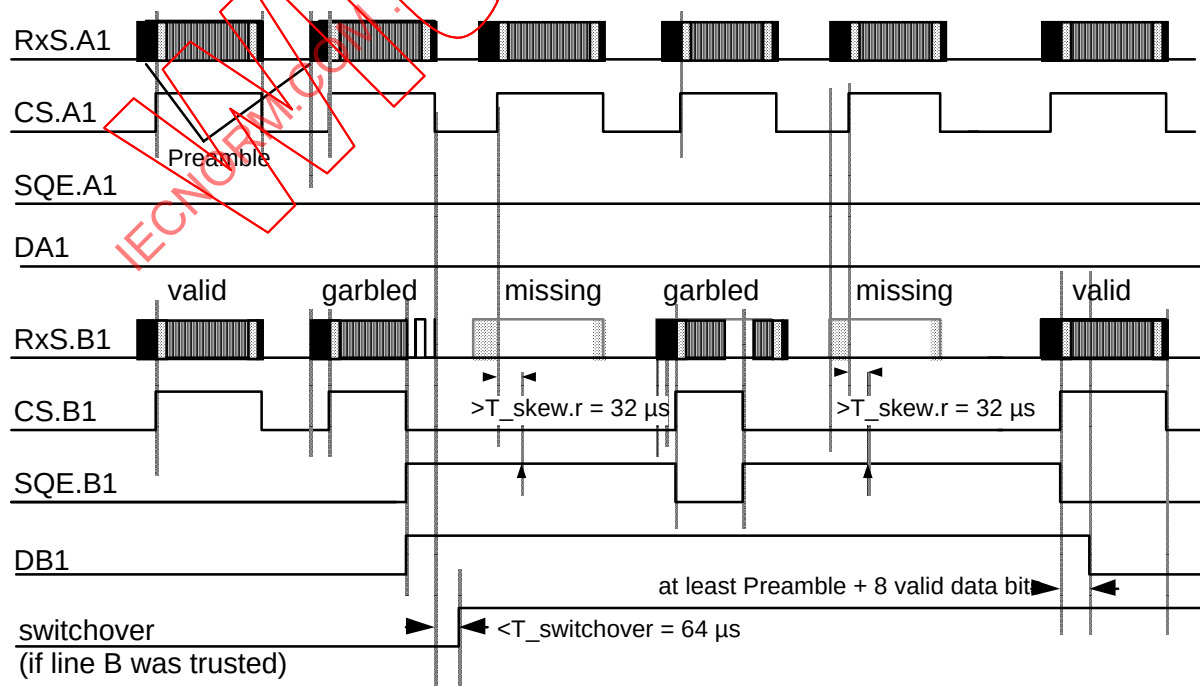


Figure 140 - Line_Disturbance signals

4.3.2.5 Switchover

Direction_1 and Direction_2 of an (end) node may trust different lines (for instance Line_A1 and Line_B2).

The switchover unit shall exchange the Trusted Line and the Observed Line within $T_{\text{switchover}} = 64,0 \mu\text{s}$:

- a) if Line_Disturbance of the Trusted Line is asserted while Line_Disturbance of the Observed Line is not asserted;
- b) if the link layer requests it (especially if a size, FCS or protocol error occurs).

4.3.2.6 MAU report

The MAU shall report to the link management:

- a) over which line it received a frame;
- b) each assertion of Line_Disturbance on either line and direction,
- c) the state of the four Line_Disturbance signals (DA1, DA2, DB1, DB2) for the Node Report (see 4.7.2.2).

NOTE If a line is totally disabled (or not connected), Line_Disturbance will toggle once after occurrence of the failure. However, if the line is interrupted, Line_Disturbance can toggle after almost every frame depending on the location of the interruption.

4.3.3 Line Unit interface

The Line Unit Interface defines the signals entering and leaving a Line Unit.

This interface can remain internal to a node, but it is recommended to make it available for testing. This interface is not covered by conformance testing.

The following specification eases interfacing and defines signals used in other Subclauses of this Standard.

If exposed, the Line Unit Interface shall consist of modem signals as defined by the UIT-T recommendation V.24, for each transceiver separately, and of additional control signals, as specified in Table 61.

Table 61 - Signals of the Line Unit Interface

Name	Designation	Clause of Rec. V.24 Of ITU-T	Direction	Meaning
GND	Signal Ground	102	-	common return
TxD	Transmitter Data	103	to Line Unit	Frame_Data, with no Preamble or End Delimiter, supplied as NRZ signal, clocked by TxC.
RxD	Receiver Data	104	from Line Unit	NRZ sequence, excluding Preamble and End Delimiter, clocked by RxC.
RTS	Request To Send	105	to Line Unit	commands to send the Preamble; clearing this signal commands the generation of the End Delimiter.
CTS	Clear To Send	106	from Line Unit	signals that the Preamble has been transmitted and requests data to follow.
TxC	Transmitter Clock	114	from Line Unit	generated by the transceiver to clock in the TxD data to send.
RxC	Receiver Clock	115	from Line Unit	generated by the decoder to clock out the received RxD data.
SQE	Signal_Quality_Error	Not V.24	from Line Unit	as specified in 4.3.1.5.2
CS	Carrier_Sense	Not V.24	from Line Unit	as specified in 4.3.1.5.1
Kx	Switch control signals	Not V.24	to Line Unit	define at least the two basic switch settings: End Setting and Intermediate Setting, for both lines. In addition, switch control signals may isolate a transceiver from the line or connect it to the line.

4.4 Frames and telegrams

4.4.1 Frame_Data format

The Frame_Data format shall conform to the HDLC format defined by ISO/IEC 3309, as shown in Figure 141.

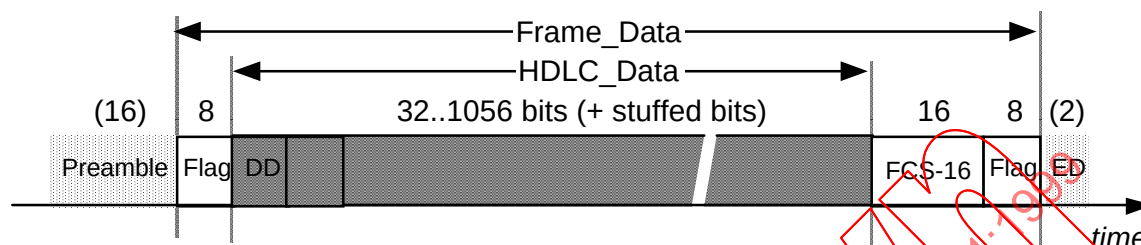


Figure 141 - HDLC Frame structure

A frame shall start with a single flag consisting of one '0' bit followed by six contiguous '1' bits and closed by one '0' bit, as specified in ISO/IEC 3309.

The opening flag shall be followed by the HDLC Data, consisting of at least 32 bits and at most of 1056 bits (not counting the stuffed bits foreseen by ISO/IEC 3309).

The HDLC Data shall have (as a restriction to ISO/IEC 3309) an integer number of octets.

The first octet of the HDLC Data shall be the 8-bit Destination Device specified in 4.5.1.

The second octet of the HDLC Data shall not be interpreted as the control field of ISO/IEC 3309.

The HDLC Data shall be followed by an error detecting code, which shall be the 16-bit Frame Check Sequence (FCS) built as ISO/IEC 3309 prescribes.

The frame shall be closed by a single closing flag identical to the opening flag.

A closing flag may not be used as the opening flag for the next frame.

The transmitter shall not transmit the 'idle' or 'cancel' sequences of ISO/IEC 3309.

A frame is considered as a sequence of octets. The least significant bit of each octet shall be transmitted first, as ISO/IEC 3309 prescribes.

NOTE 1 The bit ordering convention holds uniquely for the order of transmission of bits within an octet. The two FCS octets are sent with most significant bit first due to an exception in ISO/IEC 3309.

NOTE 2 HDLC's Bit stuffing prevents flag sequences from appearing in the data between the flags: the transmitter inserts a '0' after each group of 5 consecutive '1's in the data. The receiver removes the '0' which follows a group of five '1'. Stuffed bits are invisible to the link layer, but they may increase the frame size by as much as 20%.

4.4.2 Telegram timing

4.4.2.1 Conventions

A bus segment shall be controlled by one node, called the master, which may transmit at its own pace on the bus. The other nodes are Slaves, which may transmit only when requested by the master.

An End Node is considered to be the master of the Auxiliary Channel.

The master function and the slave function within a node are distinct.

The bus traffic shall consist of paired frames, called a telegram, consisting of a Master Frame sent by the master, to which a slave responds within a specified time by a Slave Frame.

The interframe spacing shall be measured from the last transition of an End Delimiter to the middle transition of the start bit of the Preamble.

EXAMPLE - The timing of a telegram is shown in Figure 142.

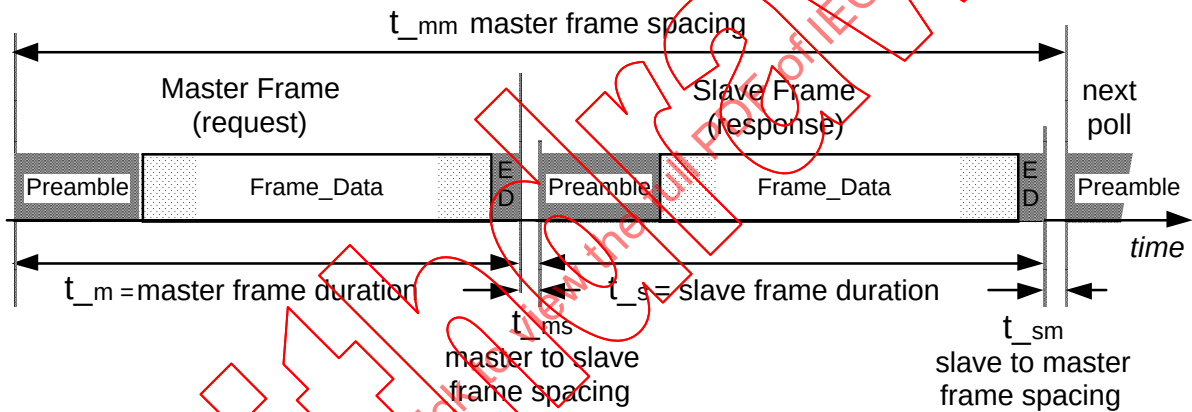


Figure 142 - Telegram timing

4.4.2.2 Computation of the Reply Delay

The reply delay T_{reply} is, for a given bus, the maximum delay which may appear between the end of a Master Frame and the beginning of the Slave Frame sent in response to it, measured at the master.

The reply delay consists of the sum of propagation delays, decoding and access delay.

T_{reply} is a configuration parameter which tells the master how long it shall wait before sending the next Master Frame if it receives no Slave Frame.

EXAMPLE - A composition with 17 nodes (16 sections), assuming the master is at one end of the bus, is shown in Figure 143.

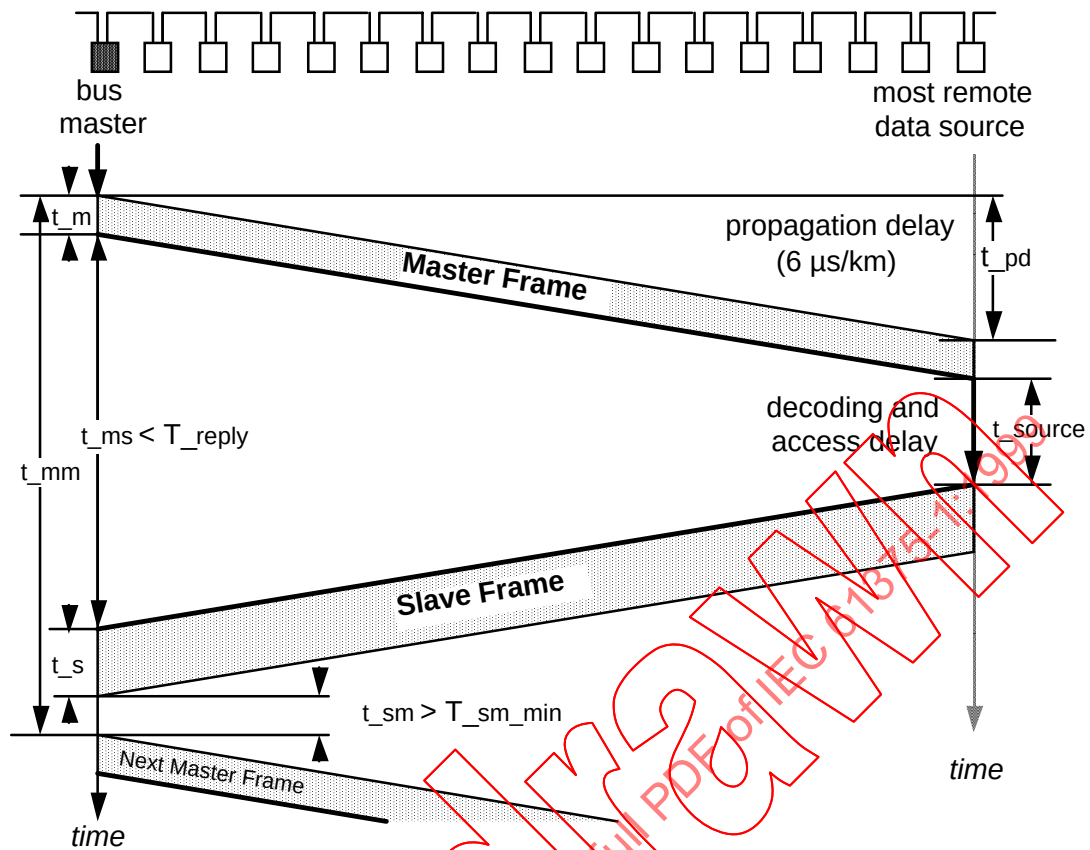


Figure 143 - Example of Interframe spacing

The worst-case reply relay for a given application, T_{reply} , is evaluated as:

$$T_{reply} [\mu s] = 2 \times T_{pd} + T_{source\_max}$$

- T_{source_max} accounts for the decoding of the Master Frame and reply at the source (see 4.4.2.4.2);
- T_{pd} is the worst case propagation delay of a frame between the End Nodes in a given application (see 4.2.2.3).

The time-outs of a frame are calculated as follows:

Bit Times	Main Channel	Auxiliary Channel
application data bits	1024 bits	16 bits
header	32 bits	32 bits
CRC	<u>16 bits</u>	<u>16 bits</u>
total bits	1072 bits	64 bits
bit stuffing (worst case x1,2)	1286 bits	77 bits
2 x flag	16 bits	16 bits
end delimiter	2 bits	2 bits
preamble maximum size	<u>32 bits</u>	<u>32 bits</u>
Total bits	1336 bits	127 bits
duration at 1,0 Mbit/s	1336,0 μs	127,0 μs
2 x propagation delay	120,0 μs	120,0 μs
response time of slave	<u>300,0 μs</u>	<u>800,0 μs</u>
Total	1756,0 μs	1047,0 μs

4.4.2.3 Collision

A collision occurs when several transmitters are simultaneously active. This situation occurs normally only between End Nodes of different bus segments which transmit simultaneously. Collisions are not distinguished from silence or non-valid frames.

4.4.2.4 Transmitted frame spacing

4.4.2.4.1 Master side

On the Main Channel, a master shall expect to receive entirely a Slave Frame in response to its Master Frame during a time-out $T_{\text{main\_max}} = 1,756 \text{ ms}$ since the end of its transmitted Master Frame, and it may start sending its next Master Frame $T_{\text{sm\_min}} = 0,064 \text{ ms}$ after the end of the received Slave Frame or after the time-out ($1,756 \text{ ms} + 0,064 \text{ ms} = 1,820 \text{ ms}$) elapsed, as Figure 144 shows:

On the Auxiliary Channel, an End Node shall expect to receive entirely a Slave Frame in response to its Master Frame during a time $T_{\text{aux\_max}} = 1,047 \text{ ms}$, and it may start sending its next Master Frame $T_{\text{ms\_min}} = 0,064 \text{ ms}$ after the end of the received Detect Response or after the time-out ($1,047 \text{ ms} + 0,064 \text{ ms} = 1,111 \text{ ms}$) elapsed.

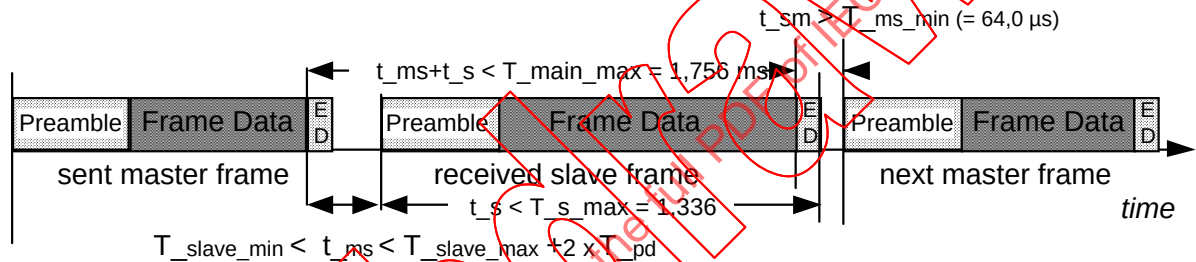


Figure 144 - Frame spacing measured at the master side

4.4.2.4.2 Slave side

The addressed slave shall start transmitting its frame, as shown in Figure 145:

- not earlier than $T_{\text{source\_min}} = 64,0 \text{ µs}$ after receiving the end of the Master Frame; and
- not later than $T_{\text{source\_max}} = 0,300 \text{ ms}$ after receiving the end of the Master Frame, except for a Detect Response for which this time is raised to $0,600 \text{ ms}$ during inauguration and $0,800 \text{ ms}$ during regular operation.

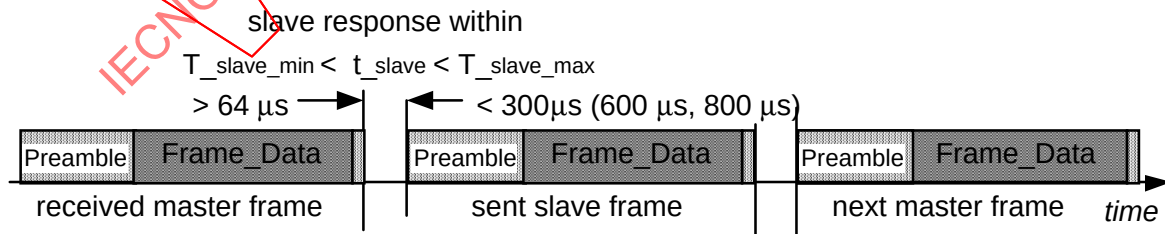


Figure 145 - Frame spacing at the slave

4.5 Link Layer Control

4.5.1 Addressing

The link layer shall use an 8-bit identifier both for the source and destination Device_Address.

The Main Channel of a node shall be addressed by a device address in the range 1 ('00000001'B) to 63 ('00111111'B) as allocated by the inauguration procedure.

The Main Channel of the master shall receive the 'master' address 1 ('00000001'B).

Device address 0 ('00000000'B) shall be the 'own' address and shall not be transmitted.

Device addresses 64 to 126 and 128 to 254 are reserved for future use.

Address 255 ('1111 1111'B) shall be the broadcast address, to which all nodes listen.

An unnamed node shall respond to address 127 ('01111111'B) over both its channels.

The Auxiliary Channel of a node shall respond to the 'unnamed' address.

A 'node' address is a named slave or a master address.

4.5.2 Frame structure

4.5.2.1 Elements of the HDLC Frame

The HDLC Data consists of the data included between the starting Flag and the check sequence, excluding stuffed bits, as shown in Figure 146:

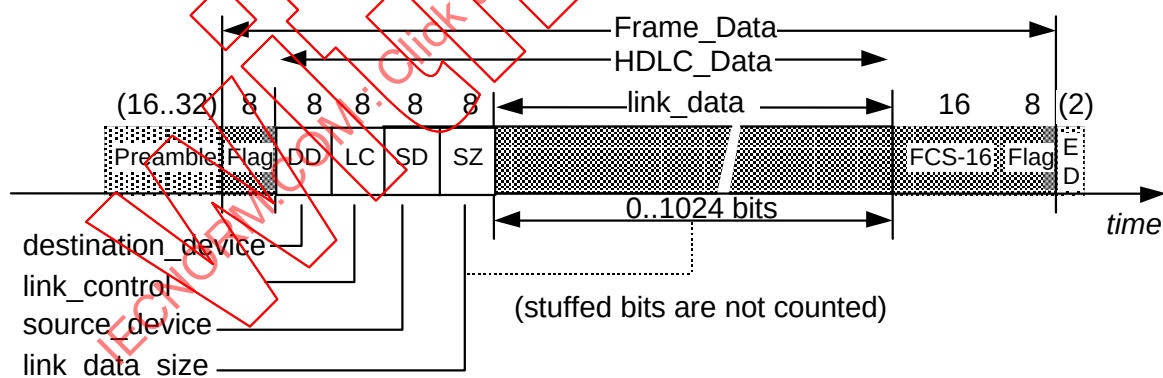


Figure 146 - HDLC Data format

The HDLC Data shall consist of the following fields, shown in Figure 147:

```

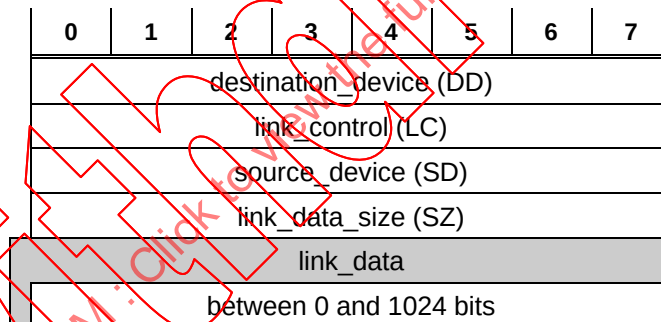
HDLC_Data ::= RECORD
{
  destination_device  UNSIGNED8      -- 8-bit address of the
                                     destination node or
                                     'broadcast' address; in a
                                     response frame, it is by
                                     default the 'master'
                                     address.

  link_control        Link_Control   -- 8-bit Link Control
  source_device       UNSIGNED8      -- 8-bit address of the source
                                     node which sends the frame;
                                     in a request frame, it is by
                                     default the 'master'
                                     address.

  link_data_size      UNSIGNED8      -- 8-bit size of the Link Data
                                     which follows the size
                                     field, expressed as an
                                     integer number of octets;
                                     it shall be zero if the Link
                                     Data field is void;

  link_data           Link_Data      -- ARRAY [link_data_size] OF
                                     WORD8
                                     between 0 and 1024 bits of
                                     data; see 4.5.3.1 for
                                     details
}

```



NOTE The aliases DD, LC, SD, SZ are used in the figures to save space.

Figure 147 - Format of HDLC Data

4.5.2.2 Link Control Field

The Link Control field distinguishes:

- Requests (Master Frames) from
- Responses (Slave Frames).

The Link Control Field distinguishes three types of telegrams:

- Process Data telegrams used to update the distributed Process data base;
- Message Data telegrams used for message transfer;
- Supervisory Data telegrams used for bus supervision and inauguration.

The Link Control field allows polled nodes to signal sporadic transmission wishes, status changes or inauguration conditions through four bits:

- a) 'A_bit': (Attention) Message Data transmission required;
- b) 'C_bit': (Change) node status change;
- c) 'I_bit': (Inhibit) inauguration not permitted
- d) 'RI_bit': (Remote Inhibit) remote inauguration not permitted.

Link Control shall be encoded into 8 bits as specified in Table 62.

Table 62 - Link Control encoding

		Encoding							
	Frame Type	0	1	2	3	4	5	6	7
Process Data and Message Data	Process_Data_Request / Process_Data_Response	M	0	A	C	I	0	0	0
	Message_Data_Request / Message_Data_Response	M	0	A	C	0	1	1	1
Supervisory Data	Detect_Request / Detect_Response	M	1	0	0	0	0	0	0
	Status_Request / Status_Response	M	1	0	RI	0	0	0	1
	SetInt_Request / SetInt_Response	M	1	0	0	0	0	1	0
	SetEnd_Request / SetEnd_Response	M	1	0	0	0	0	1	1
	Unname_Request	M	1	0	0	0	1	0	0
	Naming_Request / Naming_Response	M	1	0	0	0	1	0	1
	Topography_Request / Topography_Response	M	1	0	0	0	1	1	0
	Presence_Request / Presence_Response	M	1	0	RI	I	1	1	1

The bit combinations not specified in the Table 62 are reserved and shall be ignored.

The Link Control type is specified in textual form as:

```

Link_Control ::= RECORD
{
mq          ENUM1          -- most significant
{
SR          (0),          -- '0' in a slave response
MQ          (1)          -- '1' in a master request
}
sup         ENUM1
{
PM          (0),          -- process data or message
SP          (1)          -- supervisory data
},
ONE_OF [sup]
[PM]       RECORD
{
a_bit       BOOLEAN1,    -- '1' if A_bit set in
                        Process_Data_Response or
                        Message_Data_Response
c_bit       BOOLEAN1,    -- '1' if C_bit set in
                        Process_Data_Response or
                        Message_Data_Response
i_bit       BOOLEAN1,    -- '1' if I_bit set in
                        Process_Data_Request or
                        Process_Data_Response
pom         ENUM3
{
PROCESS_DATA (0),        -- Process_Data_Request or
                        Process_Data_Response
MESSAGE_DATA (7)        -- Message_Data_Request or
                        Message_Data_Response
}
},
[SP]       RECORD        -- Supervisory_Data_Request or
                        Supervisory_Data_Response
{
res0       WORD1 (0),    -- reserved, = 0
rem_inn    BOOLEAN1      -- '1' if 'RI_bit' set in
                        Status_Response and
                        Presence_Response
i_bit      BOOLEAN1,    -- '1' if 'I_bit' set in
                        Presence_Request or
                        Presence_Response
supervisory_type ENUM3    -- distinguishes supervisory data
{
DETECT     (0),          -- Detect_Request / Detect_Response
STATUS     (1),          -- Status_Request / Status_Response
SETINT     (2),          -- SetInt_Request / SetInt_Response
SETEND     (3),          -- SetEnd_Request / SetEnd_Response
UNNAME     (4),          -- Unname_Request
NAMING     (5),          -- Naming_Request / Naming_Response
TOPOGRAPHY (6),          -- Topography_Request or Topography_Response
PRESENCE   (7)          -- Presence_Request or Presence_Response
}
}
}
}

```

4.5.2.3 Handling of 'Attention', 'Change' and 'Inhibit'

Process_Data_Responses and Message_Data_Responses shall set the following bits to '1' to signal asynchronous events:

- a) The A_bit (Attention) shall be set as long as the Send Queue for Message Data contains frames for transmission;
- b) The C_bit (Change) shall be set to signal a change in the Node_Status, and reset when the node receives a Status Request;
- c) The I_bit (Inhibit) shall be set in the following way to inhibit inauguration:
 - as long as the application on a node inhibits inauguration, the node shall set the I_bit in all Process Data and Supervisory Data (Message_Data_Responses shall not set this bit).
 - the master shall copy the OR-combination of the I_bit in the Process_Data_Response received from every node it named to the I_bit of all Presence_Requests it sends,
 - an End Node shall copy the I_bit received in a Presence_Request to the I_bit of its Detect Requests and Detect Responses and to the I_bit of its Presence_Responses;
- d) The RI_bit shall be set in the following way to inhibit inauguration:
 - an End Node shall insert the I_bit which it reads from the Detect Response of a remote composition to the RI_bit of its Presence_Responses and Status Responses.

4.5.2.4 Size, FCS and protocol errors

The following applies only to frames which have been received via the own address or via the broadcast address:

A receiver shall ignore a frame and consider the line over which the ignored frame was received as disturbed:

- a) if its Frame Check Sequence is erroneous;
- b) if its length does not match the length expressed in the 'link_data_size' field;
- c) if it is a Slave Frame not of the same type (Process Data, Message Data, Supervisory Data) as the Master Frame which precedes it.

The Auxiliary Channel shall ignore a frame and report a protocol error if it is not of one of: Detect Request, Detect Response or Naming Request.

The Main Channel shall ignore a frame if it is a Detect Request or a Detect Response.

The master shall ignore a second Slave Frame coming in response to a Master Frame (collision without disturbance).

4.5.3 Telegram formats and protocols

4.5.3.1 Link Data field

The link layer control distinguishes Process Data, Message Data and Supervisory Data.

To consider addressing constraints, the definition of HDLC Data includes the Link Header:

```

HDLCD_Data ::= RECORD
{
  ONE_OF [link_control.sup]           -- depends on link_control
  {
    [PM] ONE_OF [link_control.pom]    -- Process Data or Message Data
    {
      [PROCESS_DATA]
      ONE_OF [link_control.mq]       -- Request or Response
      {
        [MQ] Process_Data_Request,
        [SR] Process_Data_Response
      }
      [MESSAGE_DATA]
      ONE_OF [link_control.mq]       -- Request or Response
      {
        [MQ] Message_Data_Request,
        [SR] Message_Data_Response
      }
    },
    [SP] Supervisory Data
  }
}
  
```

The first four octets of the Link Data form the Link Header and have the same format and meaning in all frames.

4.5.3.2 Process Data

4.5.3.2.1 Action

A master shall request the transmission of Process Data from another node (or from itself) or (in option) send Process Data to one node, by sending a Process Data Request, to which the addressed node shall respond with a Process Data Response, broadcast to all other nodes.

A Process Data telegram consists of a Process Data Request followed by a Process Data Response, as shown in Figure 148.

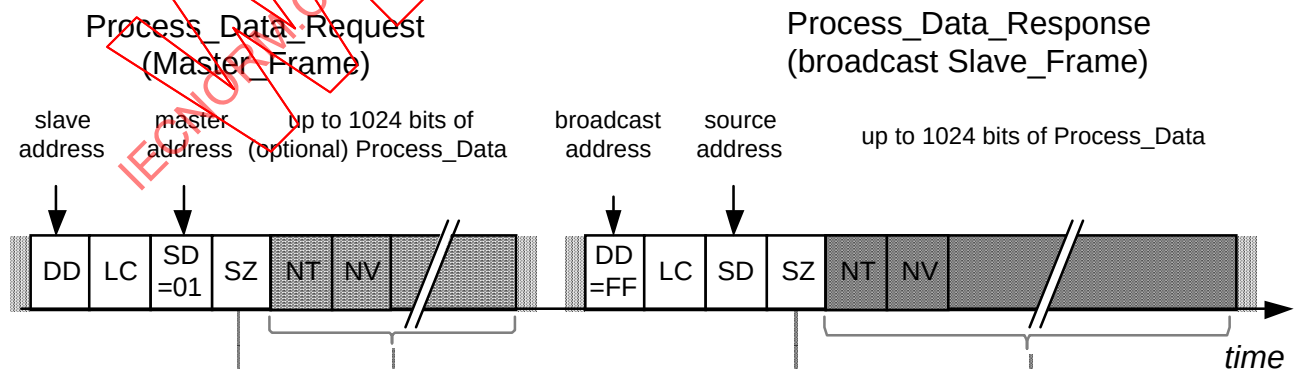


Figure 148 - Process Data telegram

4.5.3.2.2 Process Data Request

A Process Data Request shall have the following format, as shown in Figure 149:

```

Process_Data_Request ::= RECORD
{
    destination_device      UNSIGNED8      -- node address or
                                           -- 'master' address for self-
                                           -- poll
    link_control             Link_Control   -- Process_Data_Request
    source_device            UNSIGNED8      -- 'master' address
    link_data_size           UNSIGNED8      -- = 0 or
                                           -- (option (0 < link_data_size
                                           -- ≤128)
    ARRAY [link_data_size] OF WORD8        -- Process Data (option)
}                                           -- contents defined by the
                                           -- application

```

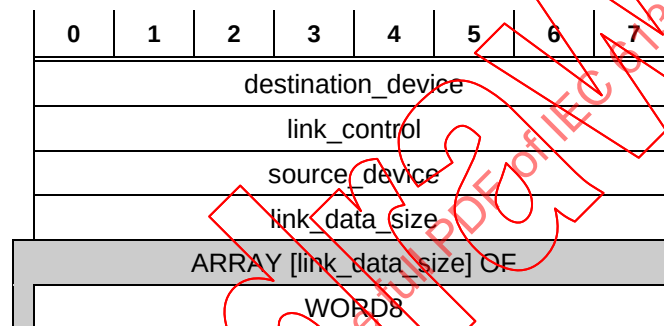


Figure 149 - Format of Process Data Request

4.5.3.2.3 Process Data Response

A Process Data Response shall have the following format, as shown in Figure 150:

```

Process_Data_Response ::= RECORD
{
    destination_device    UNSIGNED8
                           -- destination_device =
                           -- broadcast
    link_control          Link_Control
    source_device         UNSIGNED8
                           -- Process_Data_Response
                           -- node or 'master' address
    link_data_size       UNSIGNED8
                           -- (0 ≤ link_data_size ≤128)
    ARRAY [link_data_size] OF WORD8
                           -- contents defined by the
                           -- application;
                           -- it is recommended that the
                           -- two first octets be the
                           -- 'Node_Key' and that the
                           -- application checks that it
                           -- corresponds to the Node_Key
                           -- of that node it received in
                           -- the Topography.
}

```

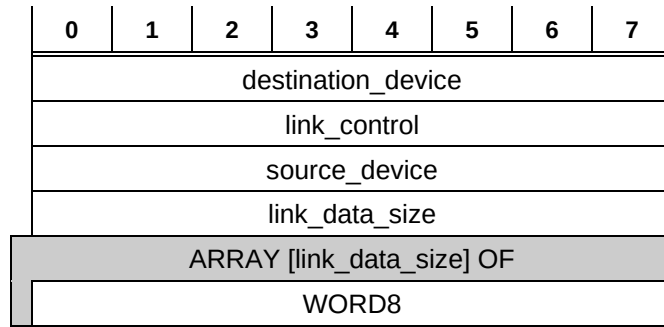


Figure 150 - Format of Process Data Response

4.5.3.3 Message Data

4.5.3.3.1 Action

A master shall request another node (or itself) to transmit Message Data with a Message_Data_Request, to which the addressed node shall respond with a Message_Data_Response, addressed to a single node or broadcast, as shown in Figure 151.

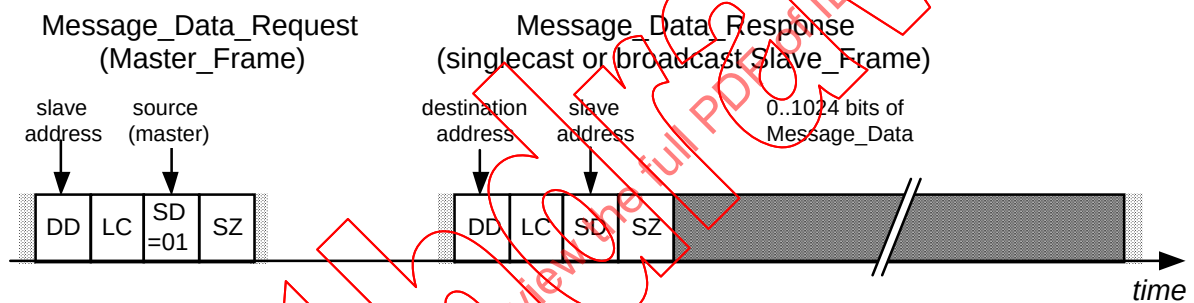


Figure 151 - Message Data telegram

NOTE The Message Data structure is specified in Clause 2.

4.5.3.3.2 Message Data Request

A Message Data Request shall have the following format, as shown by Figure 152:

```

Message_Data_Request ::= RECORD
{
  destination_device  UNSIGNED8      -- node address or
                                     -- 'master' address for self-
                                     -- poll
  link_control (= MESSAGE_DATA)      -- Message_Data_Request
  source_device       UNSIGNED8      -- 'master' address
  link_data_size      UNSIGNED8      -- =0
}                                     -- void body
    
```

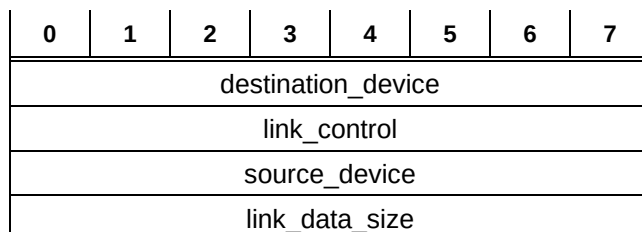


Figure 152 - Format of Message Data Request

4.5.3.3.3 Message Data Response

A Message Data Response shall have the following format, as shown by Figure 153:

Message_Data_Response ::= RECORD

```

{
  destination_device  UNSIGNED8          -- node address or
                                          broadcast address
  link_control        Link_Control       -- Message_Data_Response
  source_device       UNSIGNED8          -- node address
  link_data_size      UNSIGNED8          -- (0 ≤ link_data_size ≤128)
  ARRAY [link_data_size] OF WORD8       -- Message Data contents defined
                                          by Clause 2
}

```

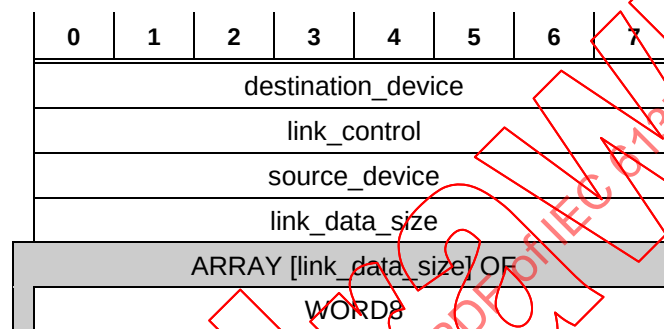


Figure 153 - Format of Message Data Response

4.5.3.4 Supervisory Data

4.5.3.4.1 Action

A master shall request Supervisory Data from a node or shall send Supervisory Data to a node with a Supervisory Data Request, to which the addressed source shall respond with a Supervisory Data Response, as shown by Figure 154.

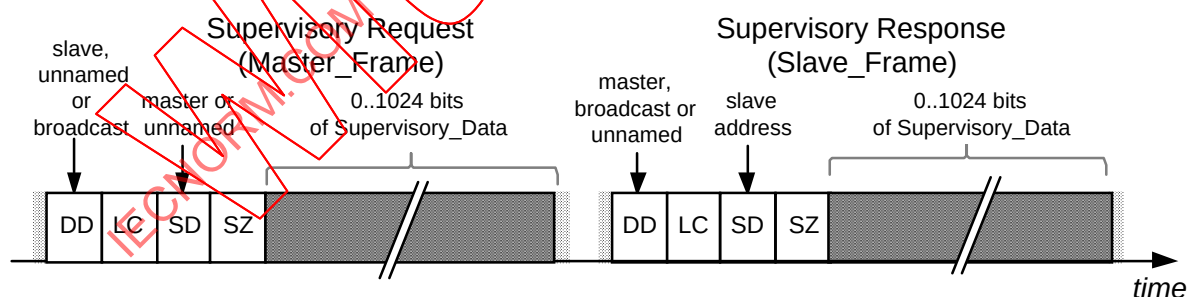


Figure 154 - Supervisory telegram

Over the Auxiliary Channel, an End Node may play the role of a master.

4.5.3.4.2 Supervisory telegram formats

Supervisory frames shall have the following format:

Supervisory_Data ::= ONE_OF [link_control.mq]

```
{
  [MQ]                Supervisory_Data_Request,
  [SR]                Supervisory_Data_Response
}
```

Supervisory_Data_Request ::= ONE_OF [link_control.supervisory_type]

```
{
  [DETECT]            Detect_Request,
  [PRESENCE]          Presence_Request,
  [STATUS]             Status_Request,
  [NAMING]             Naming_Request,
  [SETINT]             SetInt_Request,
  [SETEND]            SetEnd_Request,
  [TOPOGRAPHY]         Topography_Request,
  [UNNAME]            Unname_Request,
}
```

Supervisory_Data_Response ::= ONE_OF [link_control.supervisory_type]

```
{
  [DETECT]            Detect_Response,
  [PRESENCE]          Presence_Response,
  [STATUS]             Status_Response,
  [NAMING]             Naming_Response,
  [SETINT]             SetInt_Response,
  [SETEND]            SetEnd_Response,
  [TOPOGRAPHY]         Topography_Response
}
```

4.5.3.5 Detection telegram

4.5.3.5.1 Action

An End Node shall signal its existence to another node through a Detect Request, to which the other node (if it exists and can answer) shall respond with a Detect Response.

A detection telegram is shown in Figure 155.

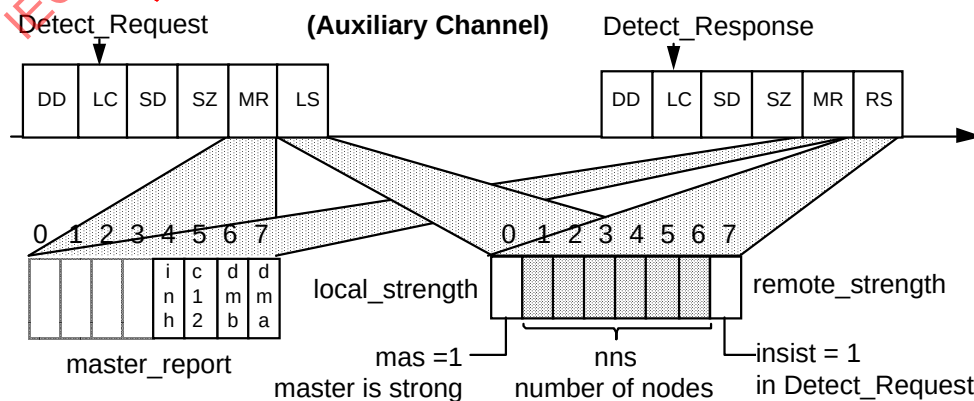


Figure 155 - Detection telegram

4.5.3.5.2 Detect_Request

Detect_Request shall have the following format, shown in Figure 156:

Detect_Request ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'unnamed' address
  link_control        Link_Control   -- Detect_Request
  source_device       UNSIGNED8      -- 'unnamed' address
  link_data_size      UNSIGNED8      -- =2
  master_report       Master_Report  -- see 4.7.2.5
  local_strength      Composition_Strength -- copy of LocStr of
                                          requesting node( 4.7.2.4)
                                          -- 'ins' is set to '1'.
}
```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							
master_report							
mas	nns					ins	

Figure 156 - Format of Detect Request

4.5.3.5.3 Detect_Response

Detect_Response shall have the following format, as shown in Figure 157:

Detect_Response ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'broadcast' address
  link_control        Link_Control   -- Detect_Response
  source_device       UNSIGNED8      -- 'unnamed' address
  link_data_size      UNSIGNED8      -- =2
  master_report       Master_Report,  -- same as in Detect_Request
                                          (for the other composition)
  remote_strength     Composition_Strength -- RemStr of responding node
                                          remote node sets 'ins' if
                                          its composition insists (
                                          4.7.2.4)
}
```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							
master_report							
mas	nns					ins	

Figure 157 - Format of Detect Response

4.5.3.6 Presence telegram

4.5.3.6.1 Action

The master shall request an End Node to signal its presence and the possible presence of another composition through a Presence Request, to which the End Node shall respond with a Presence Response, as shown in Figure 158.

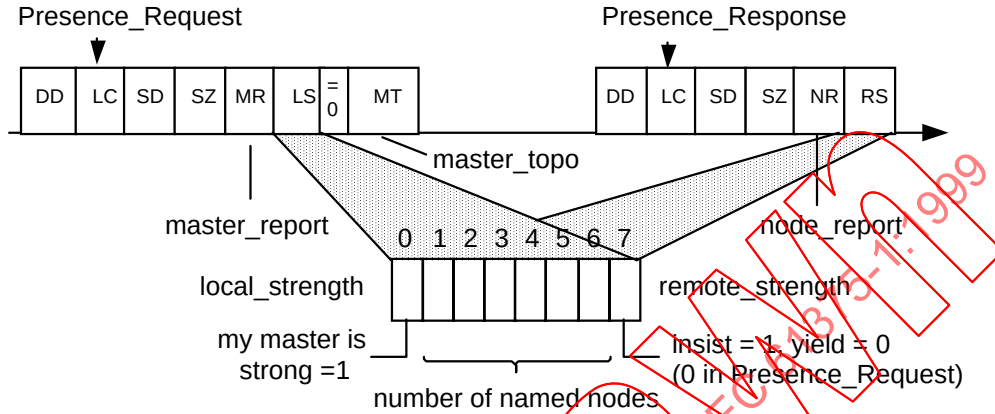


Figure 158 - Presence telegram

4.5.3.6.2 Presence Request

Presence Request shall have the following format, shown in Figure 159:

```
Presence_Request ::= RECORD
{
  destination_device  UNSIGNED8      -- address of End Node
  link_control        Link_Control    -- Presence_Request
  source_device       UNSIGNED8      -- 'master' address
  link_data_size      UNSIGNED8      -- = 4
  master_report       Master_Report   -- see 4.7.2.5
  local_strength      Composition_Strength -- copy of LocStr of master,
                                          'ins' is '0'.
  reserved1           WORD4 (=0)      -- reserved, =0
  master_topo         Master_Topo     -- see 4.7.2.7.
}
```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							
master_report							
mas	nns						ins
reserved1							
master_topo							

Figure 159 - Format of Presence Request

4.5.3.6.3 Presence Response

Presence Response shall have the following format, as shown in Figure 160:

Presence_Response ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'broadcast' address
  link_control        Link_Control   -- Presence_Response
  source_device        UNSIGNED8      -- address of End Node
  link_data_size      UNSIGNED8      -- =2
  node_report          Node_Report    -- see 4.7.2.2
  remote_strength      Composition_Strength -- copy of RemStr of End
                                         Node
                                         'ins' = '1' indicates that
                                         the other composition
                                         insists.
}
```

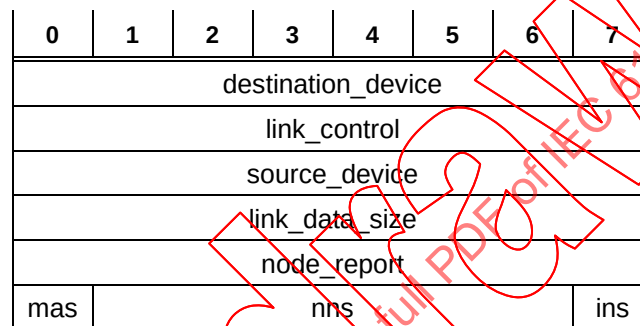


Figure 160 - Format of Presence Response

4.5.3.7 Status telegram

4.5.3.7.1 Action

The master shall request the status of a node with a Status Request, to which the slave shall respond with a Status Response, as shown in Figure 161.

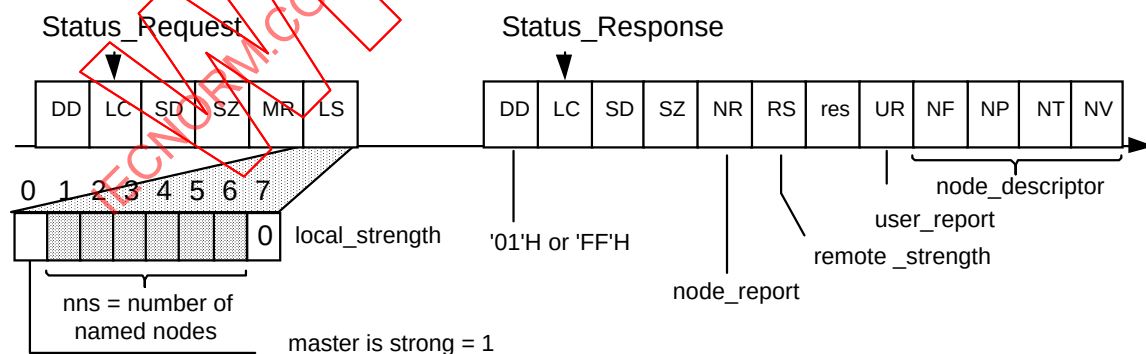


Figure 161 - Status telegram

4.5.3.7.2 Status Request

Status Request shall have the following format, shown in Figure 162:

Status_Request ::= RECORD

```
{
destination_device  UNSIGNED8      -- node address
link_control        Link_Control   -- Status_Request
source_device       UNSIGNED8      -- 'master' address
link_data_size      UNSIGNED8      -- = 2
master_report       Master_Report  -- see 4.7.2.5
local_strength      Composition_Strength -- LocStr of master, 'ins'
                                         is 0
}
```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							
master_report							
mas	nms					ins	

Figure 162 - Format of Status Request

4.5.3.7.3 Status Response

Status Response shall have the following format, shown in Figure 163:

```
Status_Response ::= RECORD
```

```
{
destination_device      UNSIGNED8      -- 'master' or 'broadcast'
                           address
link_control            Link_Control   -- Status_Response
source_device           UNSIGNED8      -- node address
link_data_size          UNSIGNED8      -- = 8
node_report             Node_Report    -- see 4.7.2.2
remote_strength         Composition_Strength  -- remote composition
                                                strength in end node.
                                                0 in intermediate node.
reserved1              WORD8 (=0)      -- reserved, =0
user_report             User_Report    -- see 4.7.2.3
node_descriptor         Node_Descriptor -- see 4.7.2.1
}
```

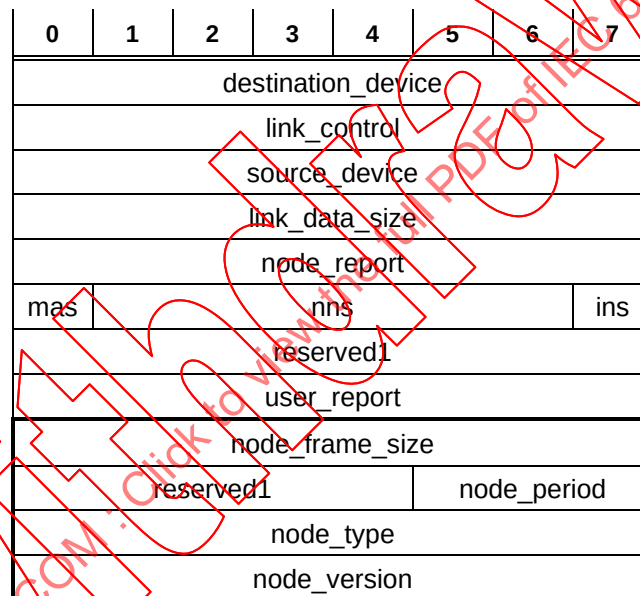


Figure 163 - Format of Status Response

4.5.3.8 Set to Intermediate telegram

4.5.3.8.1 Action

The master shall request a slave to put its switches in Intermediate Setting by sending a SetInt Request, which the slave shall acknowledge with a SetInt Response, as shown in Figure 164.



Figure 164 - Set-to-Intermediate telegram

4.5.3.8.2 SetInt Request

SetInt Request shall have the following format, shown in Figure 165:

SetInt_Request ::= RECORD

```
{
  destination_device  UNSIGNED8      -- node address
  link_control        Link_Control    -- SetInt_Request
  source_device       UNSIGNED8      -- 'master' address
  link_data_size      UNSIGNED8      -- =0
}
```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							

Figure 165 - Format of SetInt Request

4.5.3.8.3 SetInt Response

SetInt Response shall have the following format, shown in Figure 166:

SetInt_Response ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'master' address
  link_control        Link_Control    -- SetInt_Response
  source_device       UNSIGNED8      -- node address
  link_data_size      UNSIGNED8      -- =0
}
```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							

Figure 166 - Format of SetInt Response

4.5.3.9 Naming telegram

4.5.3.9.1 Action

The master shall communicate its allocated address and strength to a slave by a Naming Request, which the slave shall acknowledge with a Naming Response, as shown in Figure 167

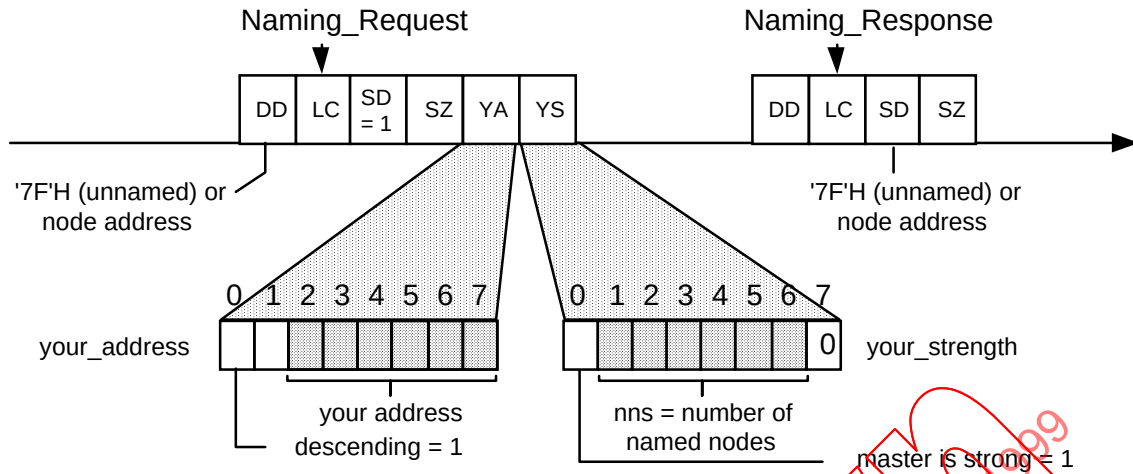


Figure 167 - Naming telegram

4.5.3.9.2 Naming Request

Naming Request shall have the following format, shown in Figure 168:

Naming_Request ::= RECORD

```

{
  destination_device  UNSIGNED8  -- node address or 'unnamed'
                                -- address
  link_control        Link_Control  -- Naming_Request
  source_device       UNSIGNED8  -- 'master' address
  link_data_size      UNSIGNED8  -- = 2
  dir1               BOOLEAN1    -- '1' if named in ascending
                                -- direction
  rsv1               WORD1       -- reserved, =0
  your_address        UNSIGNED6   -- as assigned by master
  your_strength       Composition_Strength  -- as assigned by master
                                -- 'ins' = 0
}

```

0	1	2	3	4	5	6	7
destination_device							
link_control							
source_device							
link_data_size							
dir1	rsv1	your_address					
mas	nns						ins

Figure 168 - Format of Naming Request

4.5.3.9.3 Naming Response

Naming Response shall have the following format, shown in Figure 169

Naming_Response ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'master' address or 'unnamed'
                                   address
  link_control        Link_Control   -- Naming_Request
  source_device       UNSIGNED8      -- 'node' address or 'unnamed'
                                   address (4.5.3.9)
  link_data_size      UNSIGNED8      -- =0
}
```

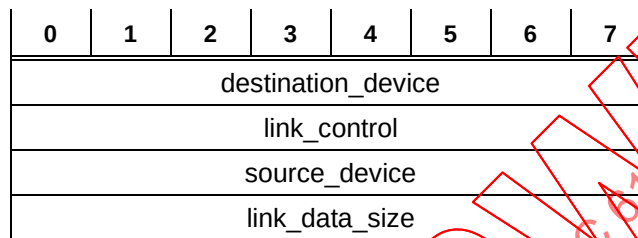


Figure 169 - Format of Naming Response

4.5.3.10 Unname telegram

4.5.3.10.1 Action

The master shall request all slaves to unname themselves by broadcasting an Unname Request, to which the slaves shall not respond, as shown in Figure 170 (there is no Unname Response).



Figure 170 - Unnaming telegram

4.5.3.10.2 Unname Request

Unname Request shall have the following format, shown in Figure 171:

Unname_Request ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'broadcast' address
  link_control        Link_Control   -- Unname_Request
  source_device       UNSIGNED8      -- 'master' address
  link_data_size      UNSIGNED8      -- =0
}
```

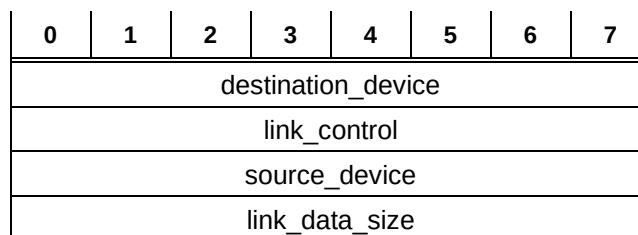


Figure 171 - Format of Unname Request

4.5.3.11 Set to End telegram

4.5.3.11.1 Action

The master shall request a slave to switch to End Setting and to adopt a new composition strength by a SetEnd Request, to which the slave shall respond with a SetEnd Response, as shown in Figure 172.

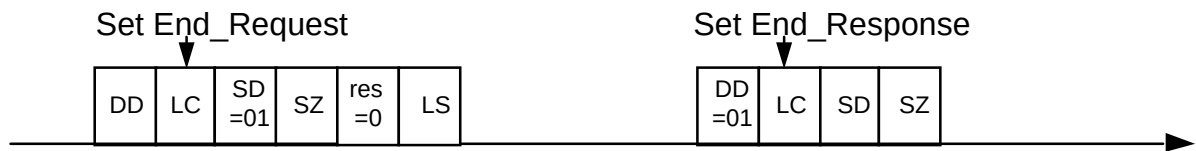


Figure 172 - Set to End telegram

4.5.3.11.2 SetEnd Request

SetEnd Request shall have the following format, shown in Figure 173:

SetEnd_Request ::= RECORD

```
{
destination_device  UNSIGNED8      -- node address
link_control        Link_Control   -- SetEnd_Request
source_device       UNSIGNED8      -- master' address
link_data_size      UNSIGNED8      -- = 2
reserved1           WORD8 (=0)     -- = 0
local_strength      Composition_Strength -- LocStr as seen by master
                                   (4.7.2.4)
}
```

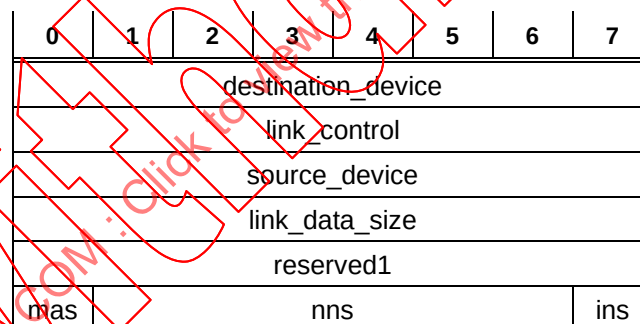


Figure 173 - Format of SetEnd Request

4.5.3.11.3 SetEnd Response

SetEnd Response shall have the following format, shown in Figure 174:

SetEnd_Response ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'master' address
  link_control        Link_Control   -- SetEnd_Response
  source_device       UNSIGNED8      -- node address
  link_data_size      UNSIGNED8      -- = 0
}
```

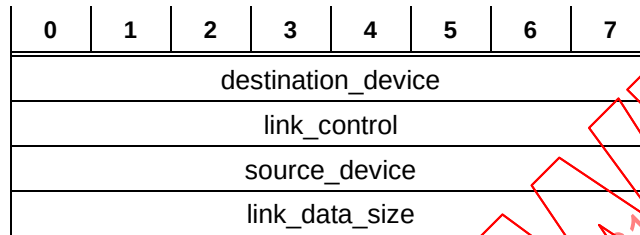


Figure 174 - Format of SetEnd Response

4.5.3.12 Topography telegram

4.5.3.12.1 Action

The master shall communicate its Topography to a slave by a Topography Request, which the slave shall acknowledge with a Topography Response, as shown in Figure 175

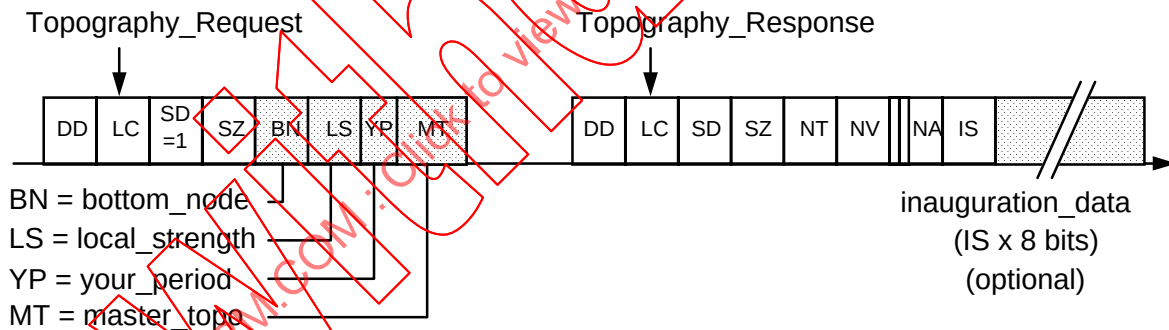


Figure 175 - Topography telegram

4.5.3.12.2 Topography Request

Topography Request shall have the following format, shown in Figure 176:

Topography_Request ::= RECORD

```

{
  destination_device  UNSIGNED8          -- node address (may be master)
  link_control        Link_Control       -- Topography_Request
  source_device       UNSIGNED8          -- 'master' address
  link_data_size      UNSIGNED8          -- = 4
  bottom_node         UNSIGNED8          -- address of End Node in
                                          Direction_1 from the master
  local_strength      Composition_Strength -- Composition strength seen
                                          by master
                                          (local_strength.ins = 0).
  your_period         UNSIGNED4          -- assigned Individual Period
                                          (4.7.2.1 and 4.6.2)
  master_topo         Master_Topo       -- see 4.7.2.7
}

```

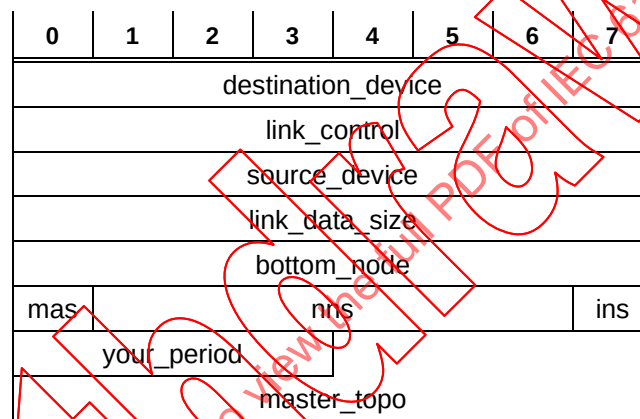


Figure 176 - Format of Topography Request

4.5.3.12.3 Topography Response

Topography Response shall have the following format, shown in Figure 177:

Topography_Response ::= RECORD

```
{
  destination_device  UNSIGNED8      -- 'broadcast' address
  link_control        Link_Control   -- Topography_Response
  source_device       UNSIGNED8      -- source node address (may be
                                     master)
  link_data_size      UNSIGNED8      -- 0 • link_data_size • 128
  node_type           Node_Type      -- first part of Node_Key
  node_version        Node_Version   -- second part of Node_Key
  sam                 BOOLEAN1       -- '1' if same direction as
                                     master
  rsv1                WORD1          -- reserved, =0
  node_address        UNSIGNED6      -- node address given by
                                     inauguration
  inaug_data_size     UNSIGNED8      -- 0 • inauguration data size ≤
                                     124 octets)
  inauguration_data   ARRAY [inaug_data_size] OF WORD8
                                     application defined
                                     inauguration data
}
```

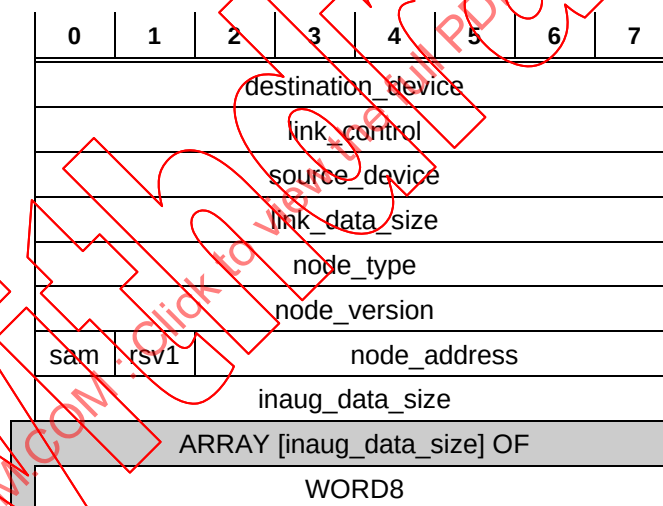


Figure 177 - Format of Topography Response

NOTE The redundancy of 'inaug_data_size' with 'link_data_size' is intentional and used for plausibility checks.

4.6 Medium allocation

4.6.1 Organisation

The following specifications apply to regular operation, when only one master is established and the bus is able to carry application data. Regular operation is entered when inauguration is finished and left when a composition change occurs.

NOTE The selection of the master among several nodes is defined in 4.7. Master selection is not part of the medium allocation described here.

4.6.1.1 Basic Period

4.6.1.1.1 Structure of the Basic Period

The master shall divide the bus activity into fixed periods, called 'Basic Periods'.

A Basic Period shall be divided into two phases, as shown in Figure 178:

- a) a Periodic Phase used to transmit periodic data and
- b) a Sporadic Phase used for transmitting:
 - Supervisory Data; and/or
 - Message Data.

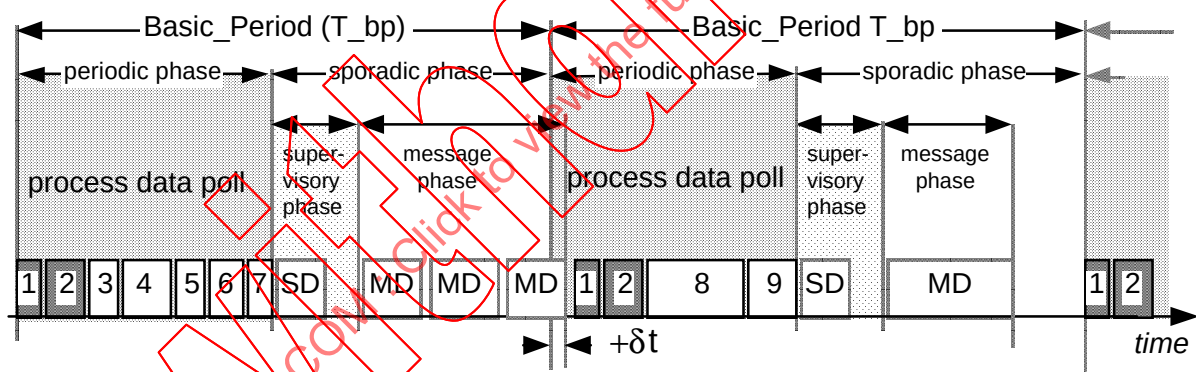


Figure 178 - Structure of the Basic Period

4.6.1.1.2 Duration of the Basic Period

The duration of the Basic Period shall be $T_{bp} = 25,0 \text{ ms} \pm 1,0 \text{ ms}$.

A master may not issue a Message Data Request or Supervisory Data Request after the beginning of the Basic Period, as long as the Periodic Phase is not completed.

NOTE This means that the start of the Periodic Phase may be delayed by the time δt it takes to transmit the longest possible Message Data or Supervisory Data frame including poll, which is about 2,0 ms. The next Basic Period is expected to start at the scheduled time if there is no other sporadic data under way.

4.6.2 Periodic Phase

4.6.2.1 Individual Period

The interval between two subsequent polls of the same node is called the Individual Period, T_{ip} .

The Individual Period shall be a multiple of the Basic Period T_{bp} , such that $T_{ip} = (2n \times T_{bp})$.

NOTE The Individual Period is defined by the application for each node. Each node announces its desired Individual Period ($node\_period$) and Slave Frame size ($node\_frame\_size$) in its Node Descriptor (see 4.7.2.1) during Inauguration.

The longest Individual Period of any node on the bus defines the Macro Period.

4.6.2.2 Periodic List

The Periodic List is the list of all nodes which are polled in each Basic Period of a Macro Period. It also defines the time left for the Sporadic Phase in each Basic Period.

The master shall configure the Periodic List on the basis of the Individual Period desired by each node ($node\_period$) and the size of the Process Data ($node\_frame\_size$) received from each node during Inauguration.

The master shall spread the polls evenly over the Basic Periods, so as to leave 40% of each Basic Period for the Sporadic Phase.

If the Periodic Phase occupies more than 60% of the Basic Period, the Individual Periods of the nodes with the longest period shall be doubled until the Periodic Phase takes less than 60% of the Basic Period, averaged over the Macro Period.

If this action is not sufficient, the period of the nodes with the second longest Node Period shall be doubled, and so on until the period of the nodes with the shortest period is doubled, if this is needed.

The Individual Period chosen for each node shall be communicated in the Topography Request to each node as $your\_period$.

4.6.2.3 Polling of the End Nodes

The master shall poll one End Node in a Basic Period, and the other End Node in the next Period, by sending a presence request frame (Presence Request), to which the End Node shall respond by broadcasting a presence response frame (Presence Response).

NOTE The Presence telegram allows all nodes to supervise the integrity of the bus and the master to detect the presence of another composition.

4.6.2.4 Error conditions and treatment

The master shall not remove from its Periodic List a node which ceases to respond, but shall continue to poll it until the next inauguration or until the node re-integrates the bus again.

NOTE 1 Failed nodes can only be removed from the composition through a new inauguration.

A node shall be able to signal to its application the disappearance of a node to which it is subscribed for Process Data, which ceased to respond to three consecutive polls, and to signal its re-integration if it recovers.

NOTE 2 The sink time supervision for Process Data provides supervision of missing nodes.

The behaviour of a node in regular operation which ceases to observe the expected traffic (for instance missing End Nodes, missing master, no polling) shall be as defined in 4.7.4.9.3.

4.6.3 Sporadic phase

4.6.3.1 Event announcement

A node shall request a sporadic transmission by setting the 'A_bit' or the 'C_bit' in their Process_Data_Response or Message Data Response.

If several nodes announce a sporadic transmission during the Periodic Phase, the master shall handle these requests on a round-robin basis, so that all other requests are served before the same node is serviced again.

All Supervisory Data requests ('C_bit') shall be attended before Message Data requests ('A_bit') are attended to.

4.6.3.2 Message List

The master shall insert into its Message List the addresses of nodes which set the 'A_bit' in one of the previous Process Data or Message Data frames.

The master shall poll a node which signals a change by a Message Data Request and remove the node from the Message List when it polls the node for Message Data, except if the Message Data Response has also an A_bit set.

4.6.3.3 Supervisory List

The master shall insert into its Supervisory List the address of nodes which set the 'C_bit' in one of the previous Process Data or Message Data frames.

The master shall poll a node which signals a change by a Status Request and remove its address from the Supervisory List when it receives its Status Response.

NOTE This list is normally void. It contains the address of the End Node in case of bus lengthening, and the addresses of nodes which change their descriptor or announce a sleep request change (set asleep or cancel sleep).

4.6.3.4 Background scanning (option)

The master may poll for Message Data or Status nodes which are not included into its Message List or Supervisory List.

NOTE A node which does not send Process Data but can send Message Data is polled by background scanning.

4.7 Inauguration

4.7.1 General

4.7.1.1 Address Allocation

The inauguration procedure shall allocate each node an address, as shown in Figure 179:

- Nodes in Direction_1 from the master are sequentially numbered in descending order, starting with 63, the last named node being the bottom node;

- [illegible]

Figure 179 - Node position numbering

The application may implement different mastership strategies by ranking a node as either:

- #### 4.7.1.2.1 Strong node

A strong node exercising mastership is called a strong master.

NOTE 1 Normally, the application promotes only one node as a strong node in a composition. If there exists more than one strong node on the bus, the inauguration procedure will cause the bus to be fragmented into as many independent segments as there are strong nodes, since there may only be one master per segment.

4.7.1.2.2 Weak node

A weak node exercising mastership is called a weak master.

In a composition in which there is no strong node, a contention resolution which treats all nodes equally ensures that one and only one of the weak nodes becomes weak master and all other nodes become Slaves.

If a weak master detects the presence of a strong node or of another weak master controlling more Slaves, it will be demoted and become a slave of the other master.

A weak node, promoted by the application to a strong node, becomes a strong master and inaugurates the bus if it is not already master.

NOTE Weak nodes are allowed to operate the bus without an explicit application command. Weak nodes can overcome the failure of a master by carrying out the inauguration and assigning another (weak) node as master.

4.7.1.2.3 Slave node

A slave node is a node which the application does not allow to become master at any time.

NOTE Slave nodes may therefore not participate in recovery. This mode is useful for testing purposes or in conjunction with strong nodes.

4.7.2 Descriptors

The following data structures are used in Supervisory Data frames and in the bus management messages, for which purpose they are defined in the transfer notation. For the link layer interface, the corresponding 'C'-data types are specified in 4.8.4.

4.7.2.1 Node Descriptor

Each node shall implement a Node Descriptor to identify its characteristics.

The Node Descriptor shall be represented the following data structure, as shown in Figure 180:

```
Node_Descriptor ::= RECORD
{
  node_frame_size      UNSIGNED8,      -- 'link_data_size' of the
                                         Process_Data_Response of
                                         this node
  reserved1            WORD5 (=0)      -- reserved, =0
  node_period          UNSIGNED3       -- Individual Period requested
                                         by node, expressed as 2n
                                         multiple of the Basic Period
                                         T_bp,
                                         n = (1 .. 128):
                                         0:   1 T_bp      (25,0 ms)
                                         1:   2 T_bp      (50,0 ms)
                                         2:   4 T_bp      (100,0 ms)
                                         3:   8 T_bp      (200,0 ms)
                                         4:  16 T_bp      (400,0 ms)
                                         5:  32 T_bp      (800,0 ms)
                                         6:  64 T_bp      (1,6 s)
                                         7: 128 T_bp      (3,2 s)
  node_type            UNSIGNED8       -- first part of Node_Key
  node_version         UNSIGNED8       -- second part of Node_Key.
}
```

0	1	2	3	4	5	6	7
node_frame_size							
reserved1					node_period		
node_type							
node_version							

Figure 180 - Format of Node Descriptor

NOTE 1 The Node Descriptors are agreed between manufacturers and users for a specific application.

NOTE 2 The Node Descriptor is used in the Naming Response, Status Response and Topography Response, and as an internal variable.

NOTE 3 The UIC Node Descriptor for international coaches is specified in the UIC 556 leaflet.

NOTE 4 The application may insert the Node Key as the first two bytes of each Process Data frame, to ensure greater protection against falsification (4.8.2.2).

4.7.2.2 Node Report

Each node shall implement a Node Report to report line disturbances and changes to the composition.

Node Report shall be represented by the following structure, as shown in Figure 181.

```
Node_Report ::= BITSET8
{
  da1          (0)      -- Line_A1 disturbed, copies DA1
                        of 4.3.2.4
  da2          (1)      -- Line_A2 disturbed, copies DA2
                        of 4.3.2.4
  db1          (2)      -- Line_B1 disturbed, copies DB1
                        of 4.3.2.4
  db2          (3)      -- Line_B2 disturbed, copies DB2
                        of 4.3.2.4
  int          (4)      -- set if node in Intermediate
                        Setting,
                        reset if in End Setting.
  dsc          (5)      -- set if the Node Descriptor
                        has been changed,
                        reset if the node receives a
                        new Topography.
  slp          (6)      -- set when a node wants to
                        enter the low-power mode,
                        reset when sleep request is
                        negated.
  sam          (7)      -- set if node has the same
                        orientation as master
                        reset if it has the opposite
                        direction.
}
```

0	1	2	3	4	5	6	7
da1	da2	db1	db2	int	dsc	slp	sam

Figure 181 - Format of Node Report

4.7.2.3 User Report

Each node shall implement a User Report to transmit an application-specific octet to other applications, for instance to report application-specific disturbances.

User Report shall be represented by one octet, as shown in Figure 182

`User_Report ::= WORD8` `-- application-defined`

0	1	2	3	4	5	6	7
ur7	ur6	ur5	ur4	ur3	ur2	ur1	ur0

Figure 182 - Format of User Report

NOTE An application may set individual bits of User Report through link layer services.

4.7.2.4 Composition strength

Each node shall implement a Local Composition Strength variable (LocStr) to indicate how many nodes are present in its composition, and whether its master is strong or weak.

Each node shall implement a Remote Composition Strength variable for each Auxiliary Channel, RemStr (1) and RemStr (2), to indicate the strength of a remote composition it detected in Direction_1 or Direction_2.

NOTE These variables are only used when the corresponding channel is active. An intermediate node does not require them.

The composition strength shall be represented by the following structure, as shown in Figure 183.

`Composition_Strength ::= RECORD`
`{`
`mas` `BOOLEAN1,` `-- set if the master of this`
`composition is strong`
`reset if the master is weak.`
`nns` `UNSIGNED6,` `-- number of named nodes in the`
`composition`
`ins` `BOOLEAN1,` `-- set if the composition`
`insists in a conflict,`
`reset if the composition`
`yields`
`}`

0	1	2	3	4	5	6	7
mas	nns						ins

Figure 183 - Format of Composition Strength

To simplify strength comparisons, the local or remote composition strength shall be calculated as an unsigned octet with the value:

$$\text{RemStr or LocStr} = (\text{mas} * 128) + 2 * \text{nns} + \text{ins};$$

EXAMPLES:

RemStr = 0	no further nodes detected;
RemStr = 1	unnamed node found;
RemStr = 2	yielding lone weak master;
RemStr = 3	insisting lone weak master;
RemStr = 4	yielding weak master with one node;
RemStr = 13	insisting weak master with six nodes;
RemStr > 128	composition named by a strong master.

4.7.2.5 Master_Report

Each node shall implement the Master_Report to report disturbances of the redundancy and to allow identification of the direction of the disturbance.

Master_Report shall be represented by the following structure, as shown in Figure 184.

```
Master_Report ::=          BITSET8
{
  rsv1 (=0)                -- reserved, set to 0
  rsv2 (=0)                -- reserved, set to 0
  rsv3 (=0)                -- reserved, set to 0
  rsv4 (=0)                -- reserved, set to 0
  inh                      -- set if any node of this
                           composition inhibits
                           inauguration
  c12                      -- set if this frame is sent in
                           Direction_2 relative to this
                           node
  dmb                      -- set if Line_B of Main Channel
                           disturbed (copies the DB1 or
                           DB2 bit)
  dma                      -- set if Line_A of Main Channel
                           disturbed (copies the DA1 or
                           DA2 bit)
}
```

0	1	2	3	4	5	6	7
rsv1	rsv2	rsv3	rsv4	inh	c12	dmb	dma

Figure 184 - Master_Report

4.7.2.6 Topo Counter

Each node shall implement the Topo Counter, which is a modulo 64 counter incremented by 1 for each successive Topography received by the node. This counter shall not take the value 0, but pass directly from 63 to 1.

Topo Counter shall be represented by the following structure, as shown in Figure 185.

```
Topo_Counter ::=          UNSIGNED6          -- 0 = not used
```

0	1	2	3	4	5	6	7
x	x	topo_counter					

Figure 185 - Format of Topo Counter

NOTE The Topo Counter is used by the Real-Time Protocols to guarantee consistency of messages exchanged over the Train Bus while an inauguration is in progress.

4.7.2.7 Master Topo

Each node shall implement the Master Topo, which is a 12-bit counter incremented by 1 for each successive Topography distributed by the master.

Master Topo shall be represented by the following structure, as shown Figure 186.

Master_Topo ::= UNSIGNED12

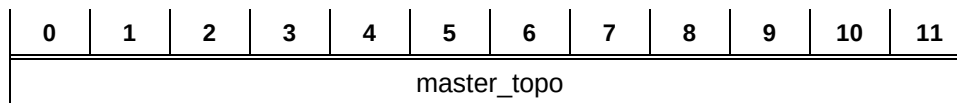


Figure 186 - Format of Master Topo

When a node becomes master for the first time, the 12-bit Master Topo shall be initialised at random.

The master shall increment the Master Topo by one each time it distributes the Topography.

NOTE This counter allows temporarily disconnected nodes to check whether they reintegrate the correct composition.

4.7.2.8 Inauguration_Counter

Each node shall implement a 16-bit Inauguration_Counter to register the number of times it went through an inauguration procedures, passing through the state of unnamed node.

NOTE This counter is used for diagnostics.

4.7.3 Detection of other compositions (informal)

4.7.3.1 Detection protocol

The End Nodes:

- detect the presence of an additional node connected to their open end(s); and
- report their own presence to that additional node.

To that effect, an End Node sends a Detect Request containing its Local Composition Strength (LocStr) towards its open end (or open ends in case of a lone master).

This Detect Request initially indicates by setting its 'insist' bit that this composition would insist in case of contention.

In all cases, an End Node (named or not) responds to a received Detect Request by a Detect Response (or another Detect Request) within the time limit of the interframe spacing (4.4.2.4).

A receiving End Node compares the strength of the remote composition (RemStr) with its own strength (LocStr):

- if the other composition is weaker than its own, it leaves its 'insist' bit set.
- if the other composition is stronger or equal to its own, it resets its 'insist' bit and yields.
- if both compositions are named by a strong master, it leaves its 'insist' bit set.

The node which receives a Presence Request adopts the strength indicated in the Presence Response.

An End Node reports in each subsequent Presence Response:

- a) the presence of another node;
- b) the strength of the remote composition,
- c) the local decision to yield or insist in case of identical strengths.

4.7.3.2 Collision avoidance rules

Since the End Nodes of two separate compositions send Detect Request asynchronously, a node could receive a garbled frame or another Detect Request when it expects a Detect Response. The following five rules reduce contention:

- a) An End Node not yet in regular operation shall send a Detect Request over its Auxiliary Channel at the latest 400,0 μ s after it receives a Naming Request or a Status Request over its Main Channel;
- b) An End Node in regular operation shall send a Detect Request over its Auxiliary Channel at the latest 400,0 μ s after it receives a Presence Request over its Main Channel, with a probability of 50% to avoid repeated collisions;
- c) A lone master shall send a Detect Request over both its Auxiliary Channels at least once every 29,0 ms, but at most once every 21,0 ms, randomising the sending within a range of $\pm 4,0$ ms around 25,0 ms to avoid repetitive collisions;
- d) An End Node, after it sent a Detect Request, shall ignore frames which are neither Detect Response nor Detect Request received during $T_{\text{detecting\_response}} = 1,047$ ms and shall ignore frames which are neither Detect Request nor Naming Requests received after that time;
- e) A master shall not cause sending of Detect Request at a rate higher than the End Node can follow. This is ensured by the protocol for Status Request, Presence Request and Naming Request.

NOTE 1 A master which is also an End Node sends a Presence Request to itself.

NOTE 2 A master which is also an End Node sends a Status Request to itself.

EXAMPLE - Figure 187 shows the timing diagram of a typical detection process.

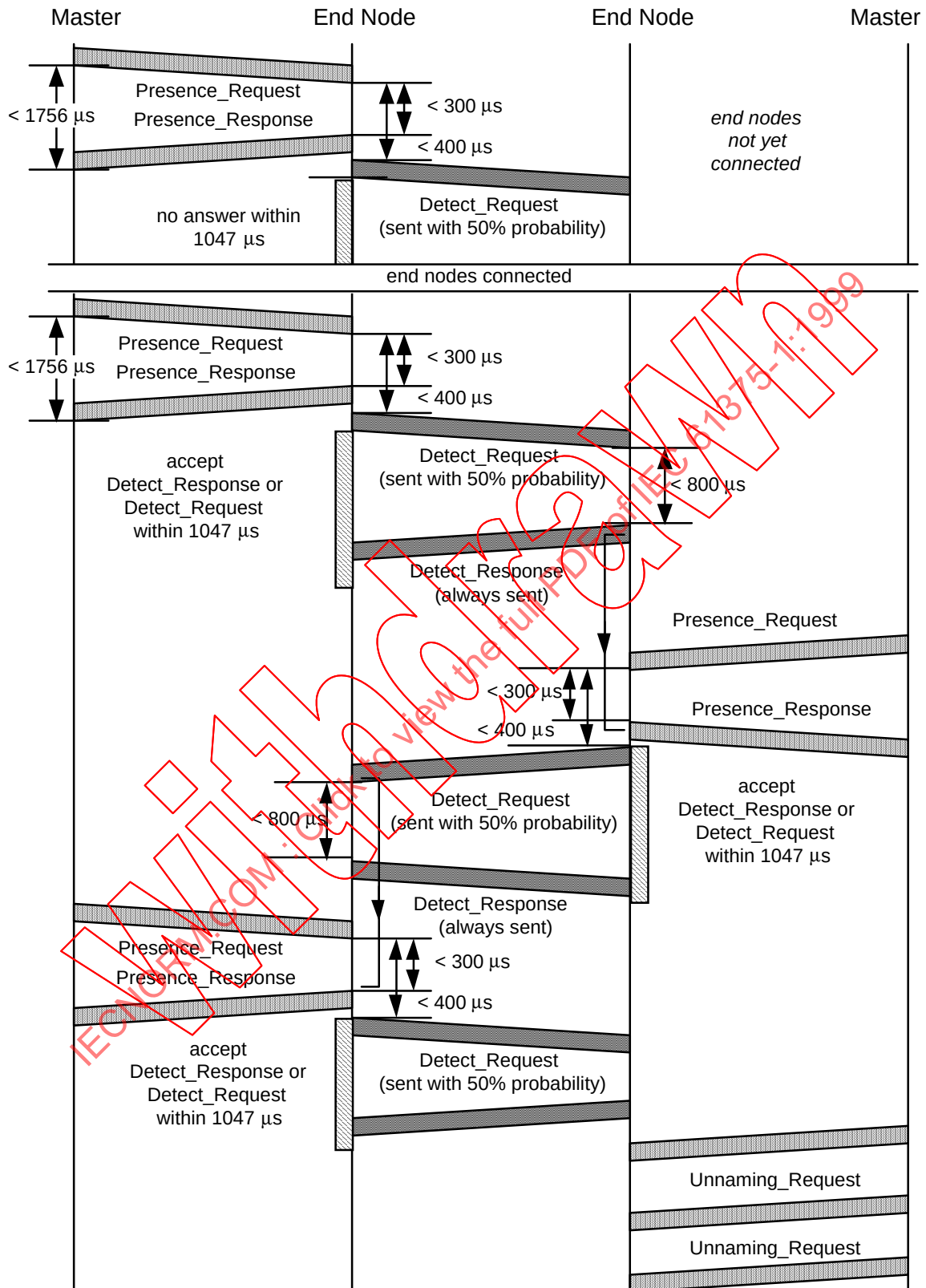


Figure 187 - Timing Diagram of detection protocol

4.7.4 State diagrams of the inauguration

4.7.4.1 Node structure

A node shall be in one of the major states shown in Figure 188. These major states subdivide into minor states that are defined in the following Subclauses.

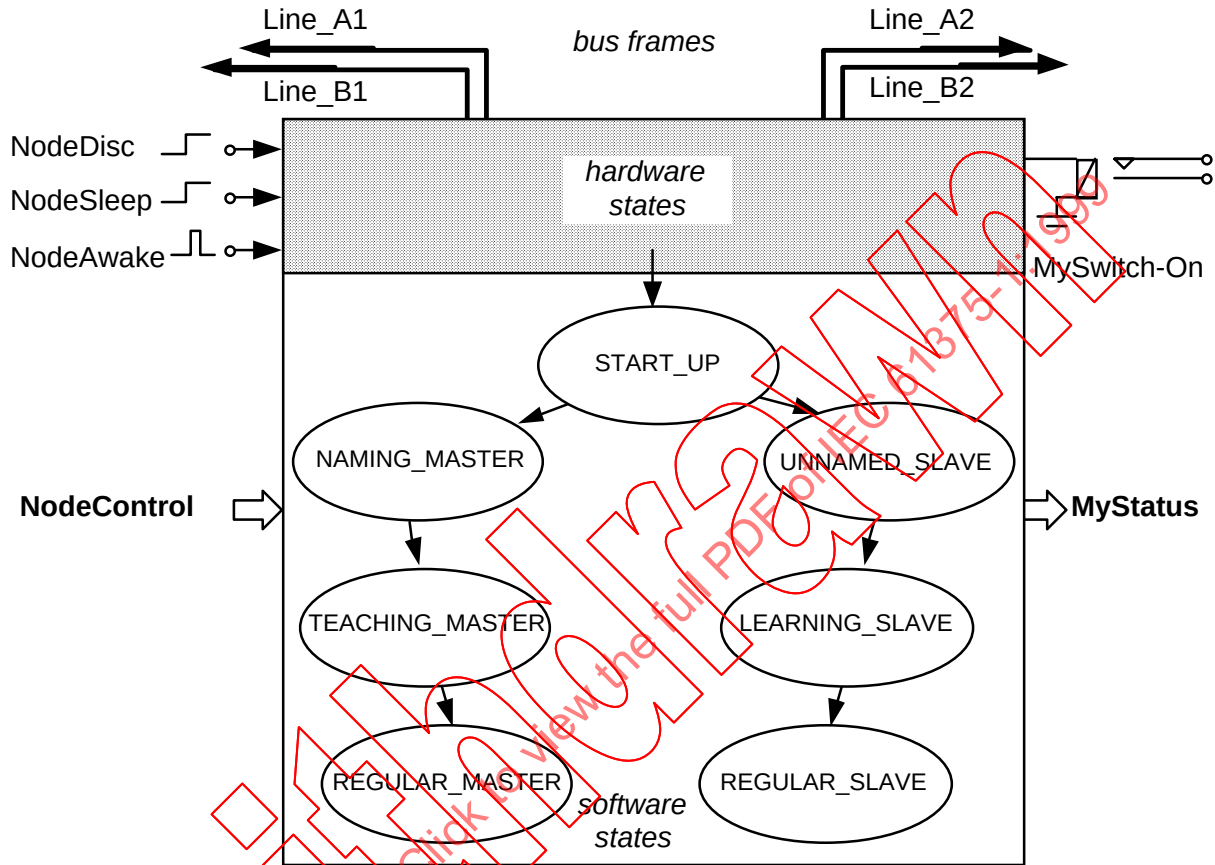


Figure 188 - Major node states and application settings

A node may be in a low-power, sleep state to save energy. Since the node is not fully powered up in sleep mode, some states and their transitions shall be implemented in the MAU hardware. These are considered as hardware states and the corresponding control and status variables are symbolised by hardware elements. The other states are considered as software states.

4.7.4.2 Inauguration process structure

Conceptually, and for the purpose of this specification, the node activity is split into three processes, as shown in Figure 189:

- a Main Process which is active on all named nodes. On a named slave, this process is attached to the direction of the master. On a master, this process is attached towards the first slave named by the master;
- two identical Auxiliary Processes, each attached to one direction. They are only active in the End Nodes in the direction of the open end of the bus.

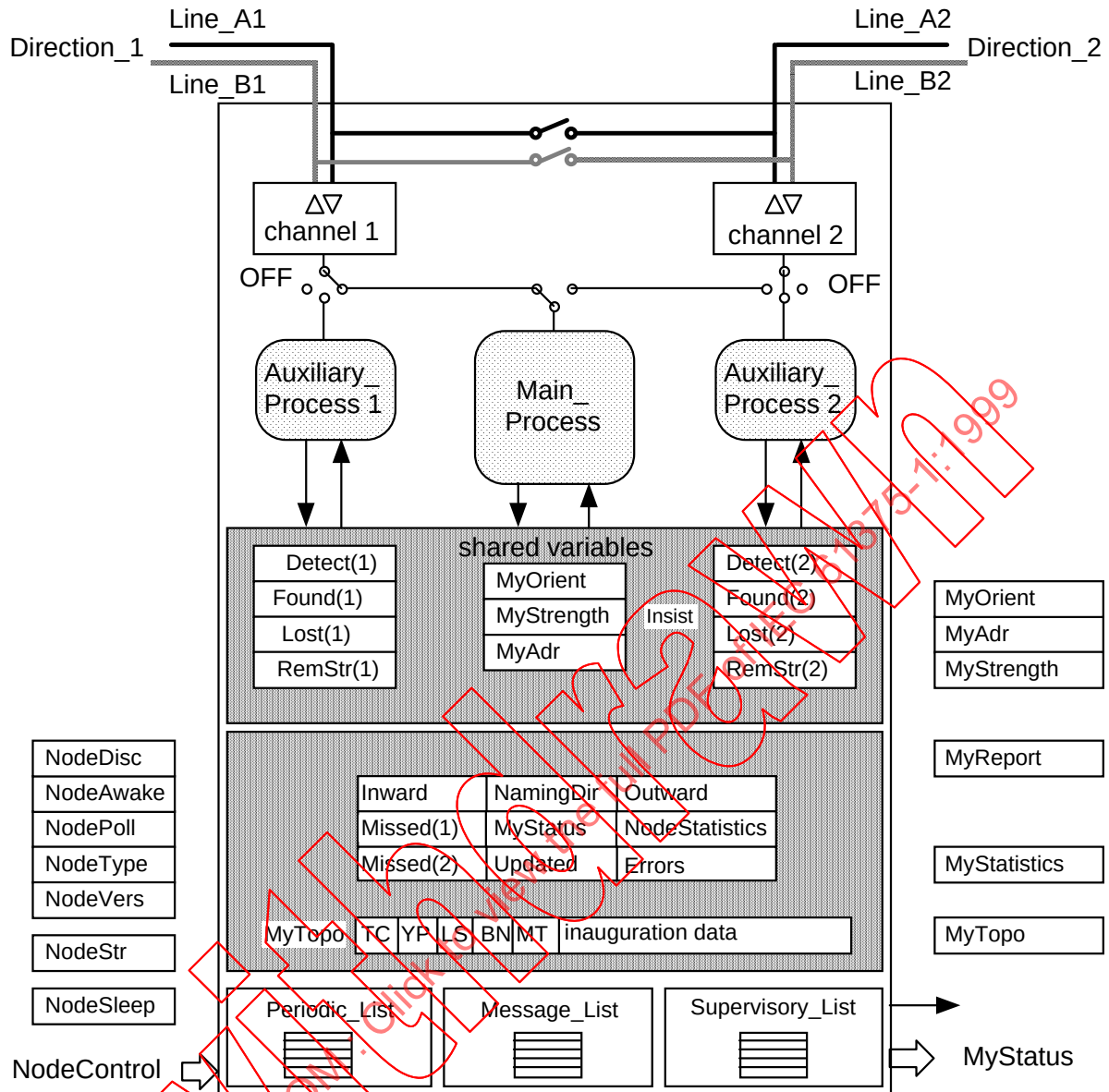


Figure 189 - Node processes (End Setting)

For the purpose of this specification, the Main Process and the Auxiliary Processes are assumed to be cyclic processes, i.e. they run at predetermined intervals (by default each Basic Period) and stop by themselves before the end of the interval. The Main Process and the Auxiliary_Process of one node run in parallel with the processes in the other nodes.

The processes control each other (and themselves):

- by sending frames over the bus;
- by starting time-outs;
- by common (polled) variables.

The transitions within a process are triggered:

- by control variables;
- by frames received over the bus;
- by elapsed time-outs.

Transitions are conditioned by common (polled) variables from the same or from other processes

The state of the node is made visible through the 'MyStatus' structure.

4.7.4.3 Specification language

The inauguration procedure is specified in the SDL/GR language [IUT-T Z.100].

To simplify the diagrams, the following conventions apply:

- a) As a restriction to SDL, there is no queuing of events, i.e. a node in a given state considers only the events which occur while the process is in that state;
- b) To simplify the diagrams, 'IF/ELSE' statements are included. Booleans take the values YES / NO or '1' / '0', or 'TRUE' / 'FALSE' indifferently.
- c) The names of the states or macros are put in all uppercase. When a macro only has one state, the state has the same name as the macro, but there is only one entry point to the macro.
- d) To each state, a timer may be associated which has the same name as the state (e.g. T_named_slave for the NAMED_SLAVE state), and which is restarted each time the state is entered.
- e) An operation returns to the state from which it has been started if no next state is specified.

4.7.4.4 Node variables

4.7.4.4.1 NodeControl

A node shall be controlled by the NodeControl data structure specified in Table 63.

Table 63- NodeControl data structure

Variable	Type	Meaning
NodeAwake	BOOLEAN1	wakes up the node from the sleep state and lets it participate in an inauguration, provided NodeSleep and NodeInhibit are both negated.
NodeDisc	BOOLEAN1	switches the node to a passive state in Intermediate Setting; it is negated by the power-up circuit; it is typically asserted when the power supply is insufficient, for instance less than 70% of nominal value; this signal may also be asserted when the node suffers an unrecoverable damage.
NodeSetUp	RECORD	command which gives a node its configuration data, and especially: NodeDescriptor and the NodeStrength (Slave/weak/strong).
NodeInhibit	BOOLEAN1	prevents the node from causing an inauguration except to recover from a failure.
NodeSleep	BOOLEAN1	sets the node to a low-power, sleep state in End Setting, but does not prevent a node from waking up if it detects activity. typically asserted when the bus should be turned off, for instance when battery charging has ceased for longer than 45 minutes

4.7.4.4.2 MyStatus

A node shall make its status available to the application through the MyStatus data structure specified in Table 64.

Table 64 - MyStatus data structure

Variable	Type	Meaning
MyTopo	RECORD	current version of the Topography as received in the latest Topography distribution; this variable is a copy of the Topography (which may be inconsistent until all nodes have received the same version).
MyOrient	ANTIVALENT2	Indicates whether the node points in the same or in the opposite direction to its master. '00'B: error, '01'B: direct, '10'B: opposite, '11'B undefined.
MyDir	BOOLEAN1	TRUE if the node has the same direction as the master, as received over Naming Request.
MyAdr	UNSIGNED6	as received over Naming Request (your_address), or 'unnamed' address for an unnamed node, or 'master' address for the master.
MyReport	NodeReport	Node Report in the format the node would send to the master in a Status Response frame (4.5.3.7).
MyStrength	Composition_Strength	Local strength of that node as received by Naming Request over the Auxiliary Channel, or by SetEnd Request, Status Request or Topography Request over the Main Channel.
MySwitchOn	BOOLEAN1	'1' if the node is to be powered up '0' switches out the node to low-power.
MyStatistics	RECORD	statistics of sent and received frames and error counts.

4.7.4.4.3 Shared variables

The Main Process and the Auxiliary Processes exchange information over shared variables.

These variables are supposed to be polled by the respective processes, i.e. there are no asynchronous signals between the Main Process and the Auxiliary Processes.

Since variables may be accessed simultaneously by several processes, variables may be locked by LOCK and UNLOCK to read and modify them consistently (for instance to avoid an unnamed node being simultaneously named from both directions).

The variables which are indexed (1,2) are specific to each Auxiliary_Process.

All variables in the MyStatus data structure are considered shared variables.

Other shared variables are specified in Table 65 .

Table 65 - Shared Variables of a node

Variable	Type	Meaning
Detect (1)	BOOLEAN1	'1' if detection is enabled in Direction_1 '0' if detection is enabled in Direction_2.
Detect (2)	BOOLEAN1	
Found (1)	BOOLEAN1	asserted: - by the Auxiliary_Process which detected another node or - by a Presence Response with a RemStr different from 0. negated: - when the node is named in that direction; - when the node is set to intermediate or - when the node becomes unnamed.
Found(2)	BOOLEAN1	
Lost(1)	INTEGER8	counters which detect the loss of an already detected, but yet unnamed node.
Lost(2)	INTEGER8	
Insist	BOOLEAN1	'1' if the end node insists in both directions, '0' if either direction detects a stronger composition.
LocStr	Composition_Strength	strength of the composition the node belongs to.
RemStr(1)	Composition_Strength	strength of another composition either detected by the own Auxiliary_Process or received in the remote_strength field of a supervisory frame.
RemStr(2)	Composition_Strength	

4.7.4.4.4 Main Process Variables

The variables listed in Table 66 are used in the Main Process. Other local variables appear in the SDL diagrams of the corresponding macros or procedures.

Table 66 - Variables of Main Process

Variable	Type	Meaning
Topo	RECORD	Topography, with one entry per named node.
TopoCounter	Topo_Counter	see 4.7.2.6
MasterTopo	Master_Topo	see 4.7.2.7
Inward	ANTIVALENT2	Direction of the master (for named slaves)
Outward	ANTIVALENT2	Direction opposite to the master (for named slaves)
NamingDir	ANTIVALENT2	Direction of naming (0 = none, 1 = Bottom, 2 = Top) for the master.
Missed(1)	UNSIGNED8	Detects the loss of an end node in each direction
Missed(2)	UNSIGNED8	
Errors	UNSIGNED8	Error counter, increased by an unfavourable finding and reset to zero by a favourable outcome of that same finding, therefore implicitly personalised for each kind of finding.
C_bit	BOOLEAN1	as long as this bit is set, all Process_Data_Responses or Message Data Responses will carry the C_bit set.

Master operation depends on the lists specified in Table 67.

Table 67 - Lists of Main Process

Variable	Type	Meaning
Periodic_List [n]	RECORD	List of the addresses to be polled for Process Data in a particular Basic Period, n being the Basic Period position in the Macro Period.
Message_List	RECORD	List of nodes to be polled for Message Data, ordered in the order as which the transmission requests have been detected (signalled by the A_bit)
Supervisory_List	RECORD	List of nodes to be polled for Supervisory Data, ordered in the order as which the transmission requests have been detected (signalled by the C_bit)

4.7.4.5 Auxiliary_Process

4.7.4.5.1 Process

As shown in Figure 190, the Auxiliary_Process detects the presence of nodes which do not belong to the composition. It is executed by nodes in End Setting only. Lone nodes have two active Auxiliary_Processes. Frame exchange takes place over the Auxiliary Channel.

The Auxiliary_Process consists of two states, a state DETECTING_RESPONSE, which is entered only if the node is named and does not yield, and a state DETECTING_RESPONSE which is entered as long as the Auxiliary_Process is active.

4.7.4.5.2 State 'DETECTING_RESPONSE'

A named node which does not yield shall send a Detect Request and expect a Detect Response or a Detect Request in the state DETECTING_RESPONSE.

It shall expect in the DETECTING_RESPONSE state:

- a) a time-out T_detecting_response
 - and go to DETECTING_REQUESTS;
- b) Detect Response or a Detect Request:
 - record the presence and the strength of the remote composition (Found (this_dir)= '1' ; Lost = 0),
 - compare the respective forces et possibly yield if the other composition is stronger, 'insist' = '0', and
 - go to DETECTING_REQUESTS;
- c) other type:
 - ignore and go to DETECTING_REQUESTS.

NOTE 1 Detect Request is the only Master Frame sent by a slave.

NOTE 2 Some situations may cause the node to shut down unconditionally.

4.7.4.5.3 State 'DETECTING_REQUESTS'

The Auxiliary_Process shall expect in the state DETECTING_REQUESTS:

- a) a time-out T_detecting:
 - If the node received no frame during T_detecting, it shall increment the 'Lost' counter of this direction and go to the 'DETECTING_RESPONSE' state;
 - If the node received no frames for MAXLOST consecutive detection periods, it shall reset the variable 'Found' and set 'RemStr (this_dir)' to zero and go to the 'DETECTING_RESPONSE' state;
- b) Detect Request:
 - register the presence of another node (Found(this_dir)= '1', Lost = 0);
 - compare the respective strengths and if the requester is stronger or has equal strength, yield if it is not strong by setting 'Insist' to '0', and
 - send a Detect Response.
- c) Naming Request:
 - if it did not previously receive a Detect Request of s stronger composition, ('Found(1)' , respectively 'Found(2)' = '0'), it shall declare the line over which this frame was received as disturbed and trust the other line, even if the other line is already disturbed.
 - otherwise it shall execute the NAMING_RESPONSE macro, which, if successful, terminates the Auxiliary_Process.

d) other type of frame:

- declare the line over which this frame was received as disturbed and trust the other line, even if the other line is already disturbed, and go to DETECTING_REQUEST without resetting T_detecting.

NOTE 1 Detect Request and Naming Request are the only types of frames that a node expects over its Auxiliary Channel. If a node receives a Naming Request without having previously received a Detect Request, this is interpreted as a protocol error.

NOTE 2 Since a Detect Request could have been garbled by a collision, the time T_detecting is randomised around a median value to avoid repeated collisions.

IECNORM.COM: Click to view the full PDF of IEC 61375-1:1999

Withdrawn

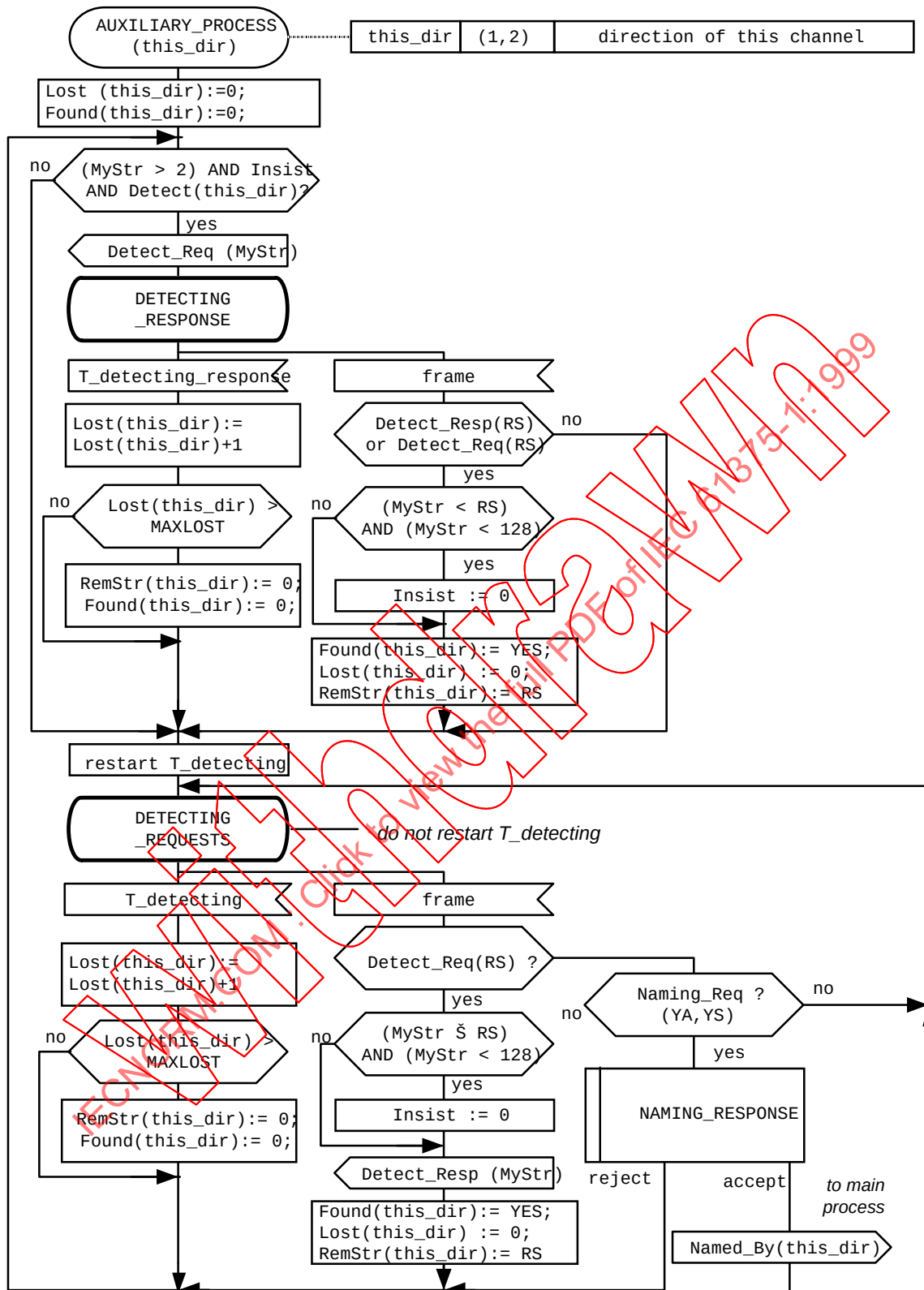


Figure 190 - AUXILIARY_PROCESS states

4.7.4.5.4 Macro 'NAMING_RESPONSE'

As shown in Figure 191, the 'NAMING_RESPONSE' macro assigns an address to a node:

- If the node is already named, it shall ignore the Naming Request. This situation should only occur when the node has been named over the other channel;
- If the node is unnamed, the node shall respond with Naming Response. It shall stop detection in that direction and allocate the channel to its Main Channel.

NOTE Checking that the node is unnamed requires an exclusive read operation of the name MyAdr since the other Auxiliary_Process may also access it.

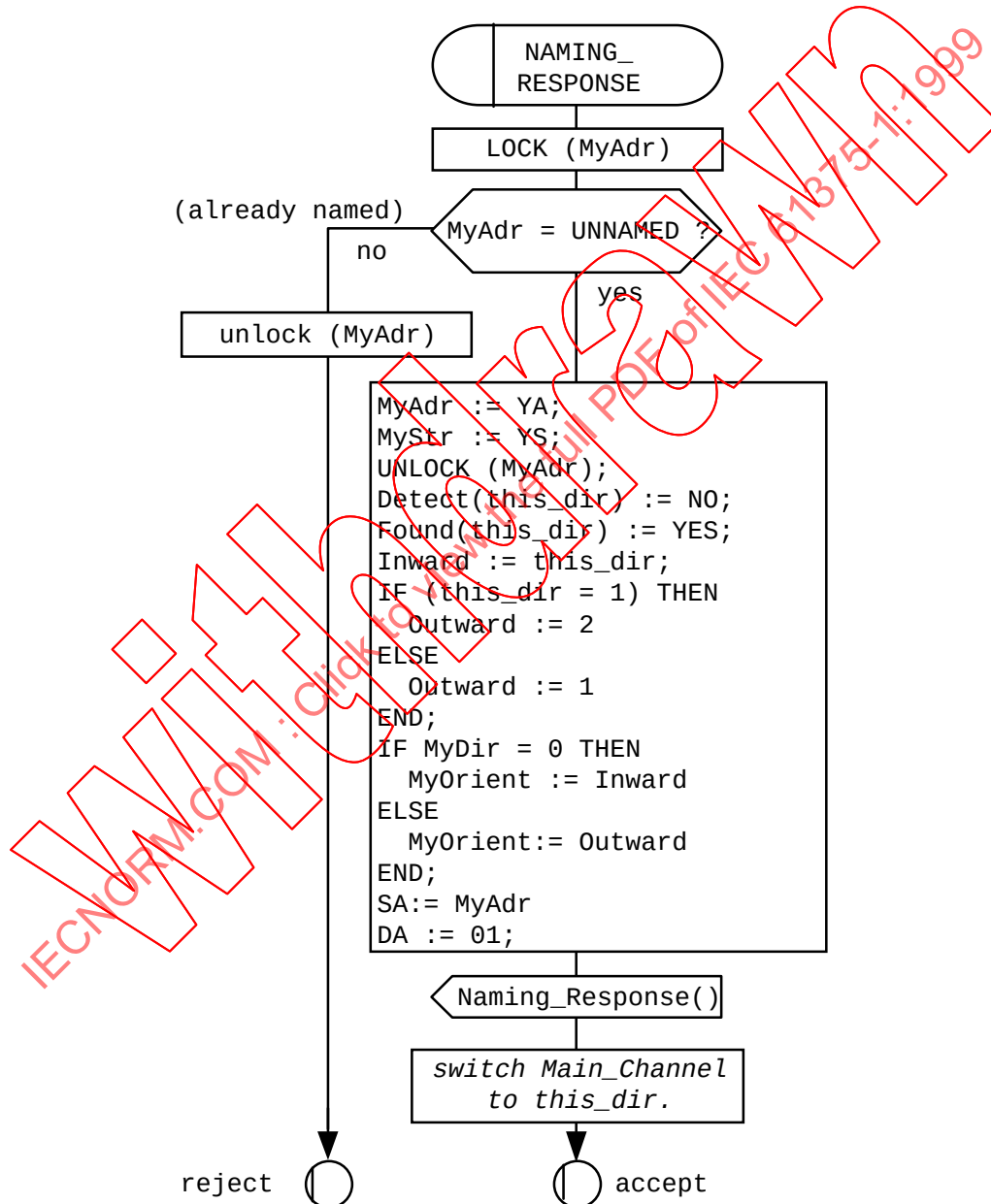


Figure 191 - NAMING_RESPONSE macro

4.7.4.6 Major states of the Main Process

As shown in Figure 192, Main Process is divided major states.

These states are subdivided into several states, represented by a macro (which appears only once) or by a procedure (which may appear several times), as described in the following Subclauses.

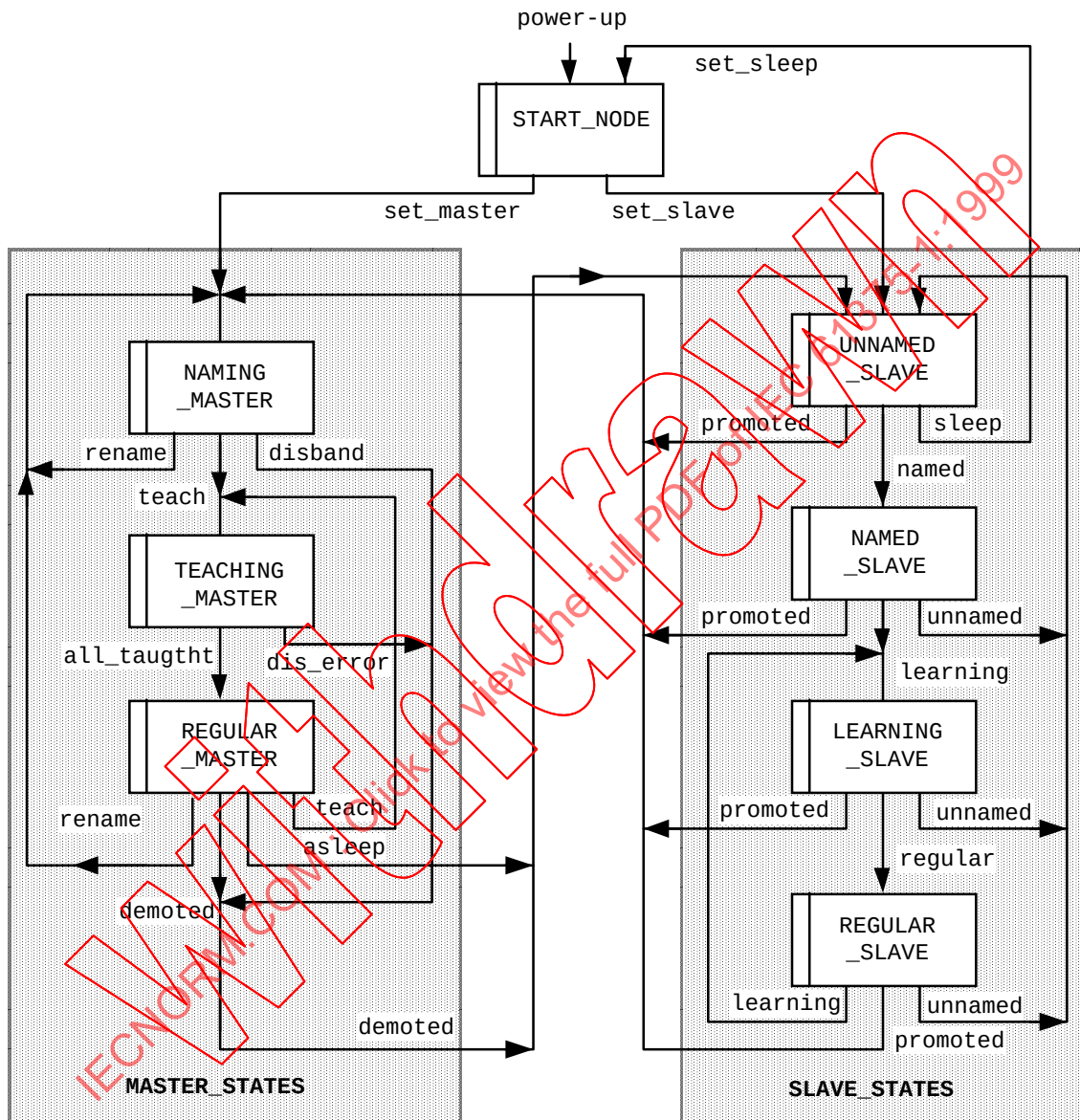


Figure 192 - States of MAIN PROCESS

The START_NODE state is described in Table 68.

Table 68 - 'START_NODE'

'START_NODE'	The node enters the state 'START_NODE' after failure, reset or power-up. It starts either in a de-energised state in which the node is passive, or from a low-power state, in which the node is waiting for a bus or application signal to restore full power. The node is then configured by the application through the NodeSetUp command. If the node has been configured as strong, it enters the NAMING_MASTER state immediately. If it is configured as weak or slave node, it enters the UNNAMED_SLAVE state.

The states in which a node operates as a master are listed in Table 69.

Table 69 - 'MASTER STATES'

'NAMING_MASTER'	Initialises the node as lone master. Sets or let node in End Setting , with both detection channels active. Starts sending detection frames in both directions to search for other nodes. When the master detects a node, it names it and updates its lists of Node Status. When it finds no more nameable nodes in either direction, the master closes the naming and goes to the state 'TEACHING_MASTER'. If a weak master finds a stronger composition, it is demoted and returns to the state 'UNNAMED_SLAVE'. It shall previously unname the nodes it named (disband). If the master finds an unrecoverable error, it restarts the procedure by going back to itself. There is no time limit in this state since this may be a legal situation (lone vehicle).
'TEACHING_MASTER'	The master distributes Topography, by requesting each node, one by one, to broadcast its descriptor and its inauguration data to all other nodes. If distribution is successful, the master enters the state 'REGULAR_MASTER'. If distribution fails, the master disbands and returns to the state 'NAMING_MASTER'.
'REGULAR_MASTER'	The master polls the nodes in regular operation. The master treats the following exceptional situations: - change of descriptor or status of any node - change of strength of the master The master leaves the state 'REGULAR_MASTER' in case of - bus shortening (End Node not any more detected) - bus lengthening (naming of additional nodes).

The states in which a node operates as a slave (SLAVE states) are listed in Table 70.

Table 70 - 'SLAVE STATES'

'UNNAMED_SLAVE'	An 'UNNAMED_SLAVE' is in End Setting , with both its Auxiliary_Processes listening (but not transmitting). It leaves this state when receiving a naming frame from a master or upon promotion by the application to the strong or weak node.
'NAMED_SLAVE'	Naming Request causes the node to switch its Main Channel toward the master. Its Auxiliary_Process on the other direction remains active. The master regularly sends a Status Request to check for further nodes on its open end. Upon detection of a further, unnamed node, the master switches the node to Intermediate Setting , which switches off the Auxiliary_Process. The node leaves this state when it receives a Topography Request or a Topography_Response, which signal the end of the naming phase.
'LEARNING_SLAVE'	The named slave receives the inauguration data from all other nodes and broadcasts its own inauguration data to all other nodes. It increments its inauguration counter by one. The node leaves this state when receiving a Presence Request, by which the master signals that it enters regular operation or any frame indicating regular operation.
'REGULAR_SLAVE'	In this state, the node is supposed to have been updated with the complete Topography. If the node is an End Node, its active Auxiliary_Process checks for bus lengthening, which it reports to the master through its Presence Response All nodes supervise the presence of both End Nodes; a node returns to the state 'UNNAMED_SLAVE' if it ceases to sense the End Nodes for three Presence Responses in sequence. If it has been updated, the node expects to be polled regularly for its Process Data and to receive the Process_Data of the other nodes. It returns to the LEARNING_SLAVE state if it receives a Topography_Response or a Topography_Request to itself, by which the master signals a composition change without address change.

In all slave states, a slave may be promoted to the strong master status and pass to the state 'NAMING_MASTER' directly.

4.7.4.7 Macro 'START_NODE'

As shown in Figure 193, this macro defines the hardware-controlled and the software-controlled states a node passes through before it becomes master or slave.

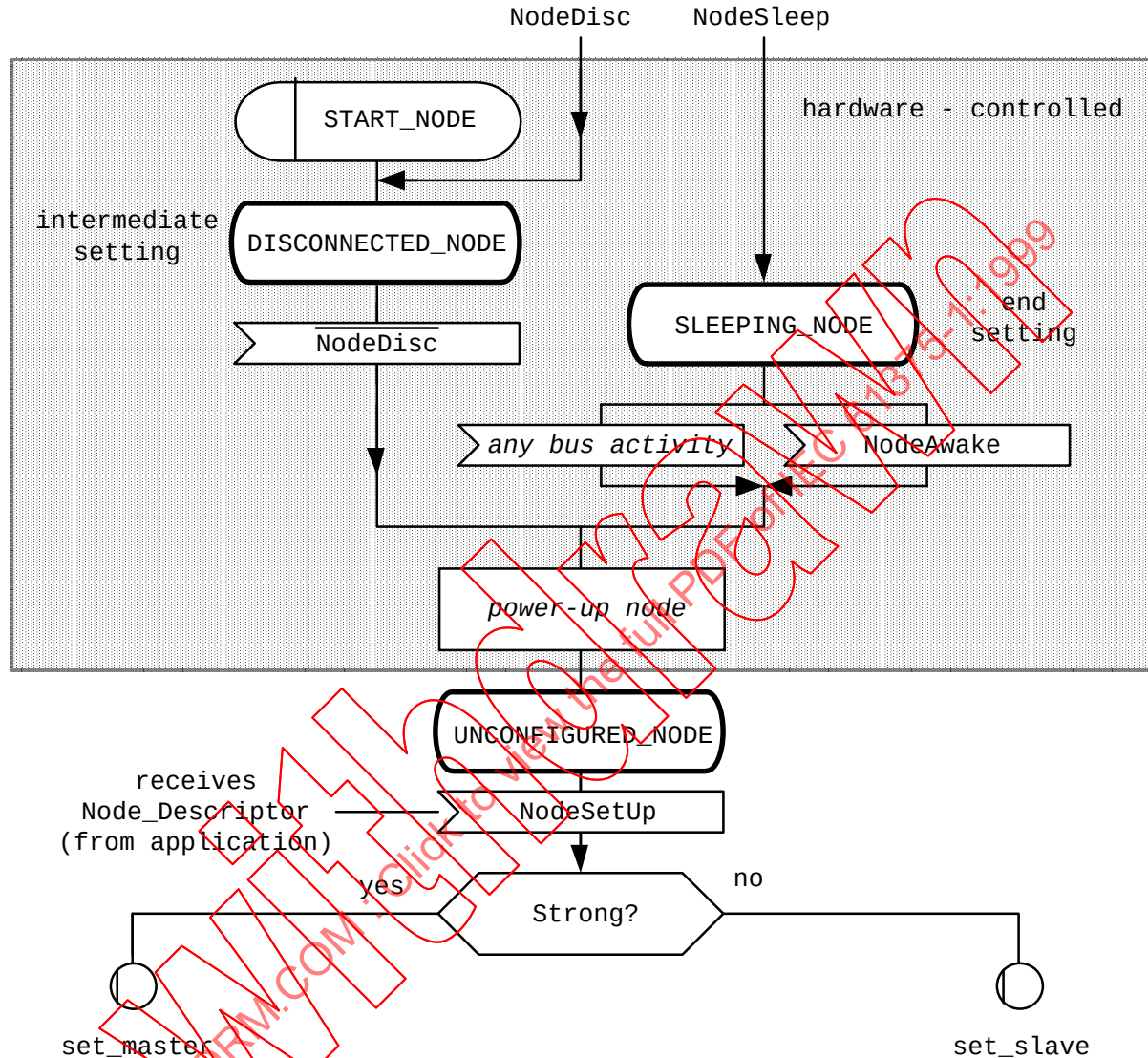


Figure 193 - Macro 'START_NODE'

4.7.4.7.1 State 'DISCONNECTED_NODE'

A node shall enter this state unconditionally when the application asserts NodeDisc, signalling that the power is about to go down or that the node suffered a severe disruption.

This is the initial state of an unpowered node. A node in the state 'DISCONNECTED_NODE' shall be in Intermediate Setting.

A node shall leave this state when the application powers the node and resets the NodeDisc signal. It shall then go to the 'UNCONFIGURED_NODE' state.

4.7.4.7.2 State 'SLEEPING_NODE'

A node shall enter that state when the NodeSleep command is asserted and the NodeDisc signal is negated.

A node shall be in End Setting, in a low-power state.

A node shall leave this state:

- a) when the application asserts the 'NodeAwake' command or,
- b) when it detects any bus activity.

When the node leaves the sleeping state, it shall assert 'MySwitchOn' to power fully up the node and go to the 'UNCONFIGURED_NODE' state. The MySwitchOn signal shall be set by hardware for all states except 'DISCONNECTED_NODE' and 'SLEEPING_NODE'

NOTE 1 Entering this state disrupts the bus. The node would be woken up again if there is activity on the bus. Therefore, the application should maintain the 'NodeSleep' signal long enough to avoid another node from awaking the node again.

NOTE 2 The bus switch is energised, since complete loss of power would bring the node to Intermediate Setting.

NOTE 3 'Any bus activity' is defined by a Carrier_Sense without a SQE signal (see 4.3.1.5), or any other mean indicating that a well-formed frame has been received over either channel.

4.7.4.7.3 State 'UNCONFIGURED_NODE'

A node shall be in End Setting and wait for its Node Descriptor and its inauguration data.

A node shall leave the UNCONFIGURED_NODE state after receiving a NodeSetUp command with a valid Node Descriptor.

It shall go to the NAMING_MASTER macro if it is a strong node, or to the UNNAMED_SLAVE macro if it is a weak node or a slave node.

4.7.4.8 Master States

4.7.4.8.1 Procedure REQUEST_RESPONSE

As shown in Figure 194, this procedure sends a request and waits for the corresponding response.

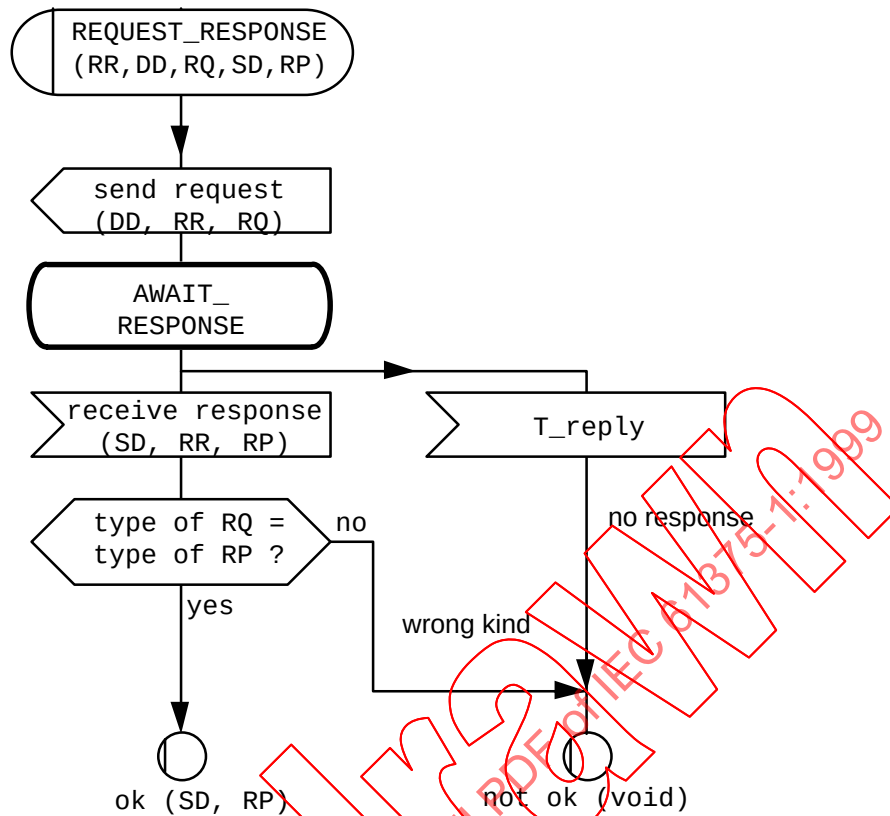


Figure 194 - Procedure REQUEST_RESPONSE

The parameters are:

- a) RR: Request/Response type, one of: {Presence, Status, Naming, Topography};
- b) DD: Destination_Device;
- c) SD: Source_Device;
- d) RP: Response parameters (depends on RR, according to the frame definition);
- e) RQ: Request parameters (depends on RR, according to the frame definition).

This procedure exits:

- with the response of the slave or
- after the T_reply time-out elapses.

4.7.4.8.2 Procedures 'SET_TO_END' and 'SET_TO_INT'

As shown in Figure 195, these two procedures set a node to Intermediate Setting, respectively to End Setting, if it is not already in that state.

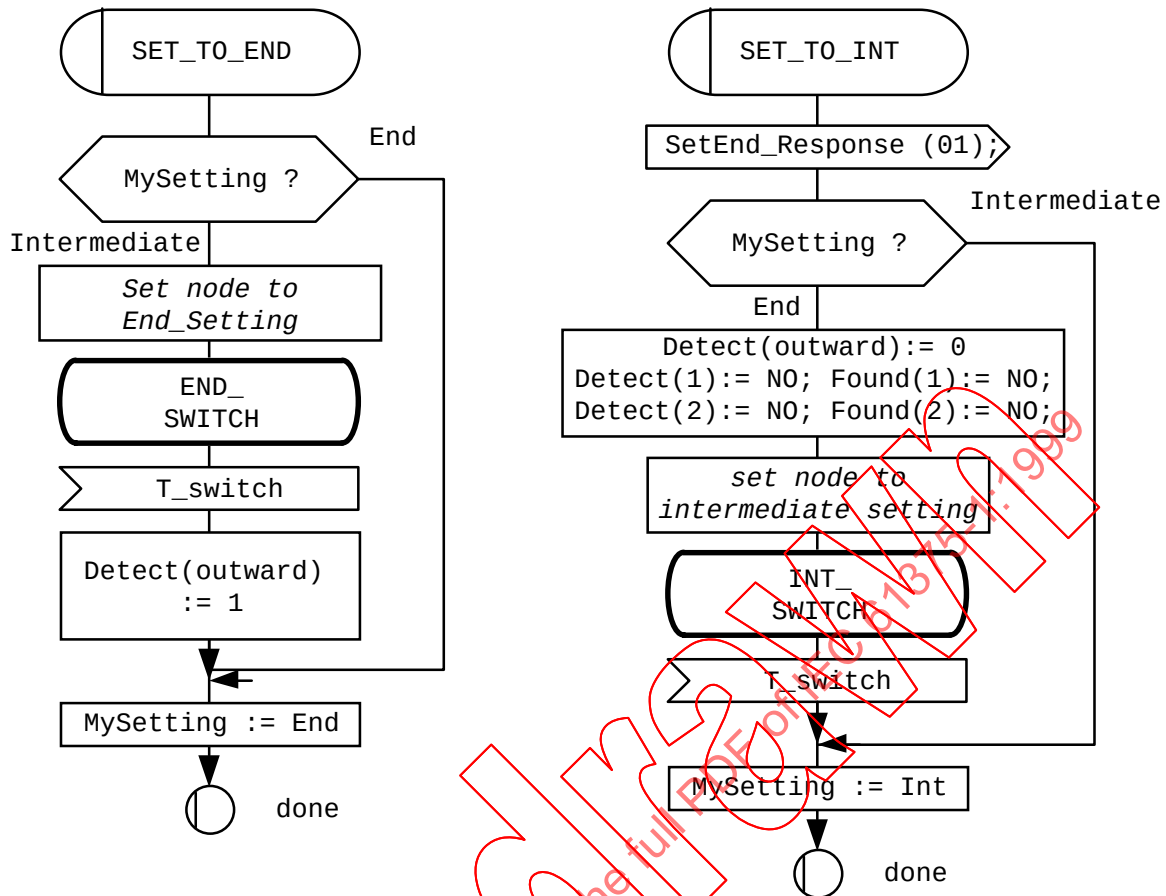


Figure 195 - Procedures 'SET_TO_INT' and 'SET_TO_END'

The Detect(1) or Detect(2) variables control respectively the starting and the stopping of the 'AUXILIARY_PROCESS'.

If the node to be set to end is the master itself, or if the node is being unnamed, both Detect(1) and Detect(2) become both '1'.

4.7.4.8.3 Macro 'INIT_MASTER'

As shown in Figure 196, this macro is entered to configure a node as master, either as a result of a promotion of a slave to strong master, or as a result of an unnamed slave becoming master by lack of bus activity.

The node first initialises itself in End Setting (if it is not already), sets up its address and strength, sets up a void Topography and enables both detection channels, which start sending Detect Requests. In this configuration, the master acts as if it would be polling itself.

After completion of the set-up, the master is ready to name other names it may find.

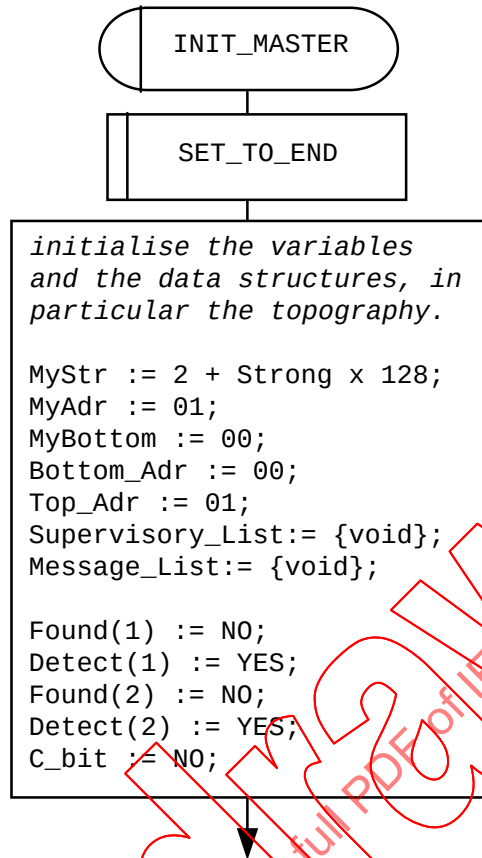


Figure 196 - Macro 'INIT_MASTER'

4.7.4.8.4 Macro 'NAMING_MASTER'

As shown in Figure 197, this macro contains the states in which the master is naming other nodes. The master enters this state by executing the INIT_MASTER macro, which initialises it.

The master shall wait in the state 'AWAIT_PERIOD' the start of a T_naming_master period. If the T_naming_master delay already elapsed, the master proceeds immediately .

The master shall execute the ASK_END macro, which asks an End Node if it discovered another composition, and acts depending on the outcome of that macro:

- if the detected composition is stronger, the master shall disband the composition by going to the UNNAMING_MASTER macro;
- if the detected composition has a weaker or an equal strength as the current composition, the master shall wait for the other composition to disband (which takes an undefined time);
- if the end node does not respond, the master shall register an error and try again later. Three consecutive errors in the same direction shall cause it to disband by executing the UNNAMING_MASTER macro;
- if the end node reports an unnamed node, the master shall name the node as new end node;
- if it detects no nameable node for three consecutive polls in the same direction, it shall go to the TEACHING_MASTER state.

NOTE 1 The master names one node per T_naming_master period.

NOTE 2 In the NAMING_MASTER state, inauguration is always permitted.

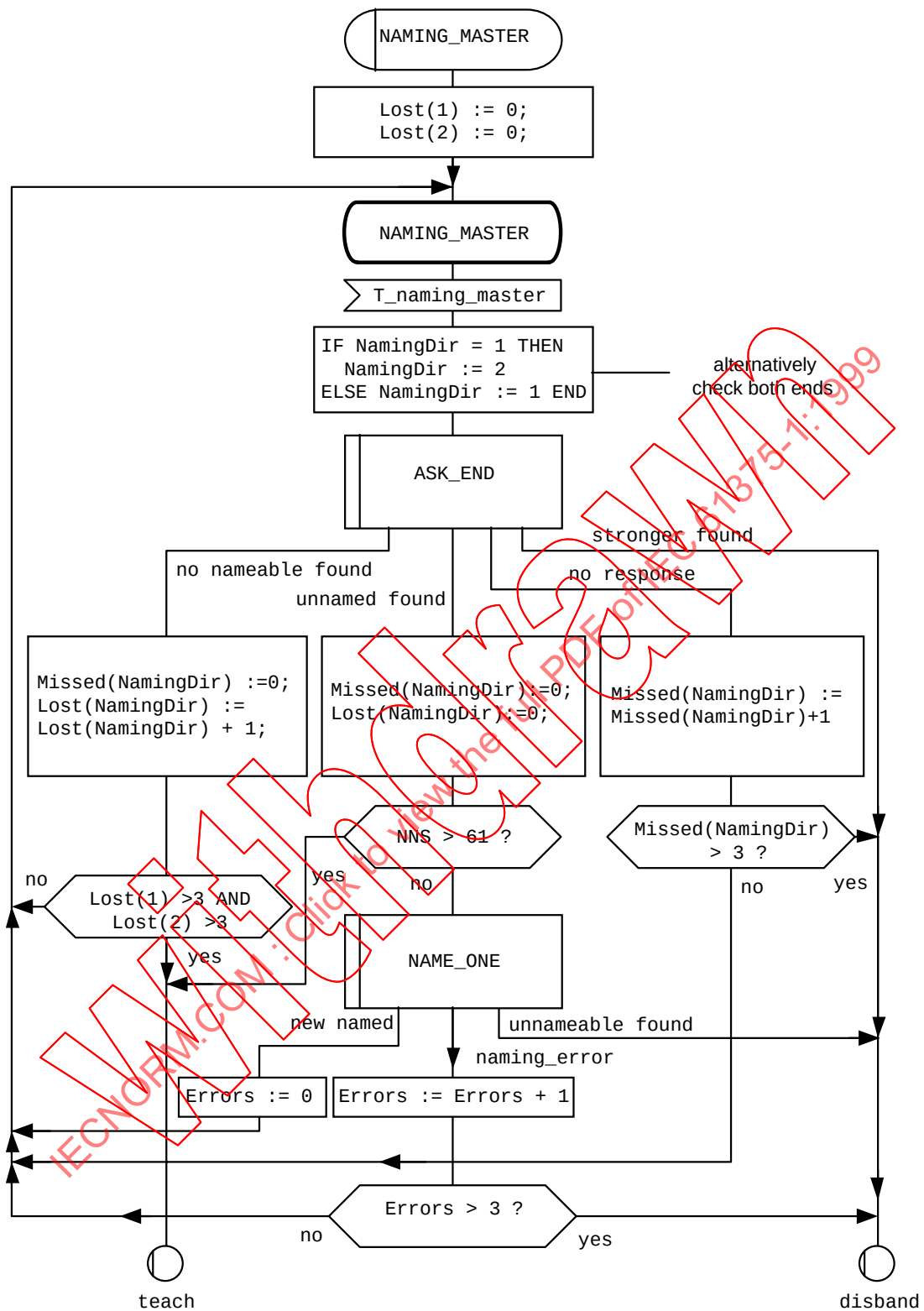


Figure 197 - Macro 'NAMING_MASTER'

4.7.4.8.5 Macro 'ASK_END'

As shown in Figure 198, this macro checks the presence of the End Node in direction 'NamingDir' and request it to report a possible remote composition. It includes implicitly the case in which the master is itself an End Node.

Depending on the detected remote strength, the master distinguishes three cases:

- no nameable node found (either no node, or an already named node);
- unnamed found (the remote node would accept naming);
- stronger found (the remote node belongs to a stronger composition).

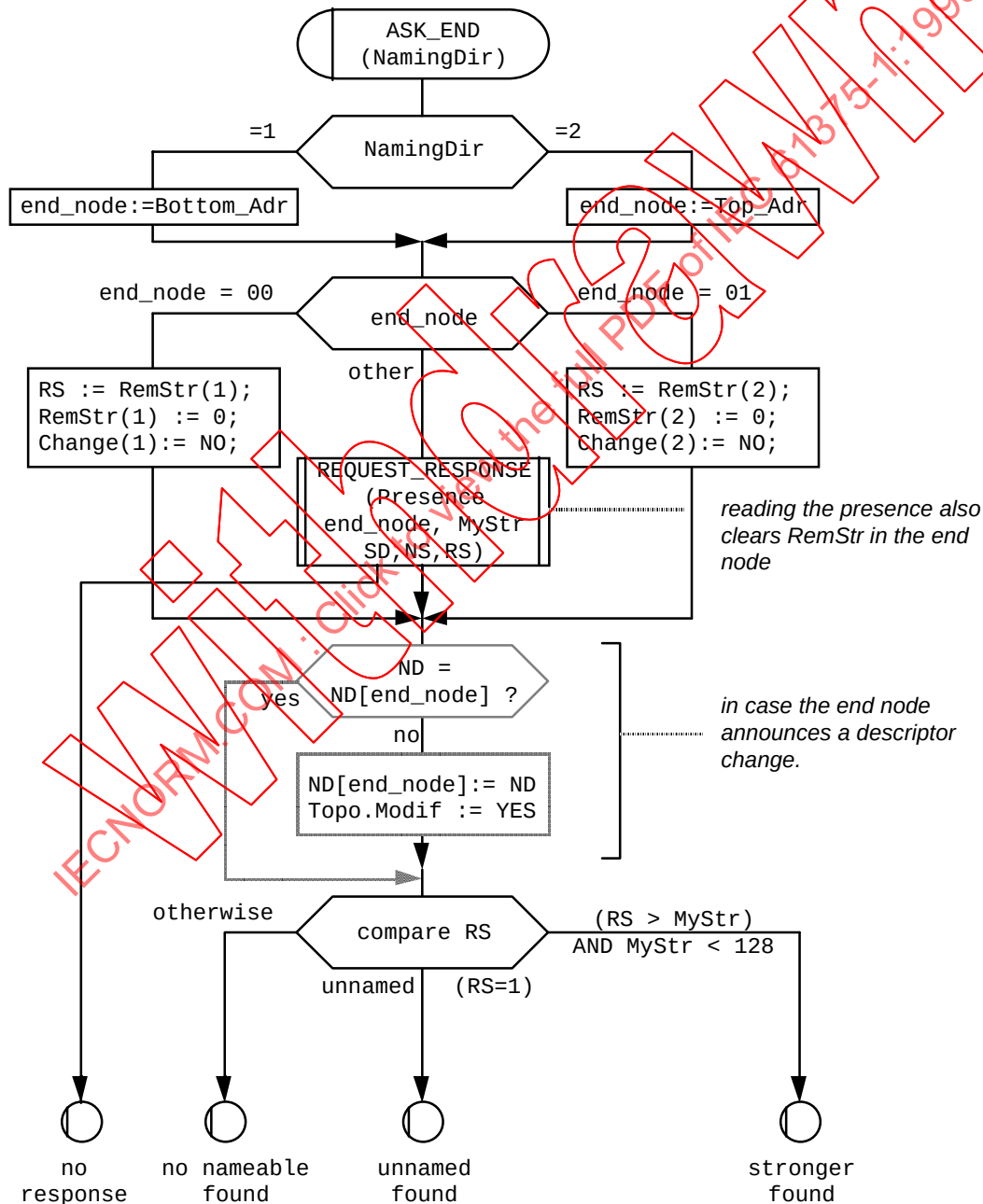


Figure 198 - Macro ASK_END

4.7.4.8.6 Procedure 'NAME_ONE'

As shown in Figure 199, this procedure of the 'NAMING_MASTER' macro is called when an End Node signalled the presence of an unnamed node.

This procedure has a parameter, the direction of naming.

If the master is itself the End Node, and it has already named nodes on the other side, it shall go to Intermediate Setting without sending a command over the bus,

Otherwise, the master shall remain in End Setting and send a SetInt Request to the End Node in the naming direction.

If it does not receive a SetInt Response, the master shall wait T_{switch} and repeat SetInt Request.

If it does not receive a SetInt Response after three trials, the master shall exit this procedure with 'naming error'.

Otherwise, after having received SetInt Response, the master shall wait a delay T_{switch} and then send to the unnamed node a Naming Request, including the address of the node as 'your_address' and its Composition Strength as 'your_strength', with the following protocol:

- a) A master shall send Naming Request with Destination_Device = 'unnamed' for the first trial.
- b) If it receives no Naming Response to that first attempt, the master shall wait a time T_{aux_main} and send again a Naming Request with Destination_Device being the allocated node address.
- c) If it receives no Naming Response to that second attempt, the master shall send again a Naming Request with Destination_Device being the 'unnamed' address.
- d) If it receives no Naming Response to that third attempt, the master shall wait a time T_{aux_main} and send again a Naming Request with Destination_Device being allocated node address.
- e) If it receives no Naming Response to that fourth attempt within T_{reply} , the master shall send again a Naming Request with Destination_Device being the 'unnamed' address.
- f) If it receives no Naming Response to that fifth attempt, the master wait a time T_{aux_main} and send again a Naming Request with Destination_Device being the allocated address.
- g) If it receives no response within T_{reply} to that sixth attempt, the master shall restore the former End Node in End Setting by sending a SetEnd Request and then wait T_{switch} for the switch to open again before leaving the NAME_ONE procedure with a status 'unnameable_found'.

Otherwise, if naming has been successful, the master shall wait T_{aux_main} and send a Status Request to the newly named node.

If the master receives the Status Response on the Trusted Line only, the Observed Line being disturbed, the master shall wait 1,2 ms before repeating the Status Request (and wait for the Status Response), repeating Status Request three times in total to assess the line quality, and communicate the result in its next Master_Report.

If it does not receive a response to three attempts of sending Status Request, the master shall exit the procedure with a status 'unnameable_found'.

Otherwise, if it received a Status Response, it shall actualise the topography, the addresses of the End Nodes and the composition strength, and exit on 'new_named'.

NOTE 1 An unnamed node is expected to receive Naming Request and to send its Naming Response over its Auxiliary Channel.

NOTE 2 The T_{aux_main} delay allows to switch from the Auxiliary Channel to the Main Channel. During this time, the bus traffic is disrupted.

NOTE 3 The newly named node accepts naming by responding with a Naming Response, or refuses naming by not responding.

NOTE 4 If the node was named but the response was lost, attempts with Destination_Device being 'unnamed' will fail. Conversely, if naming did not take place, attempts with Destination_Device being the allocated node address will fail.

NOTE 5 The named node returns the Status Response with its Node Descriptor and the Remote Strength of further nodes it may have detected.

NOTE 6 The threefold repetition of Status Request in case of disturbance allows the master to distinguish between a frame loss and the loss of one redundant line between itself and the End Node.

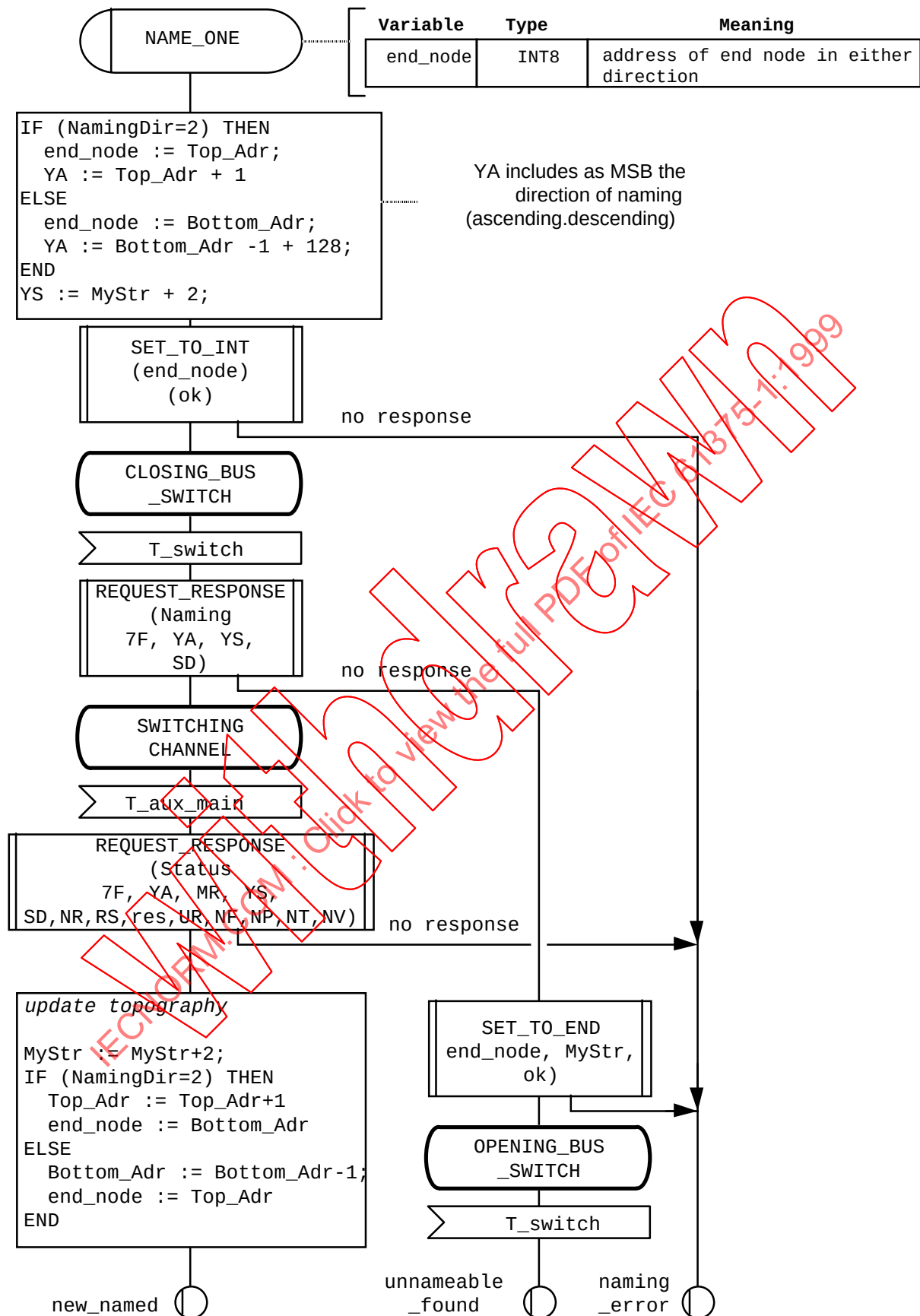


Figure 199 - Procedure NAME_ONE

4.7.4.8.7 Macro 'TEACHING_MASTER'

As shown in Figure 200, the master distributes the Topography information to all named nodes when it detects no more nodes to name or when it detects a node which signals a change in its Node Descriptor.

To this effect, it shall increment TopoCounter and MasterTopo.

The node shall send three Topography Request in sequence to all named nodes, starting with the bottom node and ending with the top node, including itself (self-poll), in order of increasing addresses.

The node shall leave this state:

- when it receives no Topography Response after having sent three times the Topography Request, in which case it shall go to the macro UNNAMING_MASTER;
- when it received from all nodes the Topography Response, in which case it shall go to the macro REGULAR_MASTER.

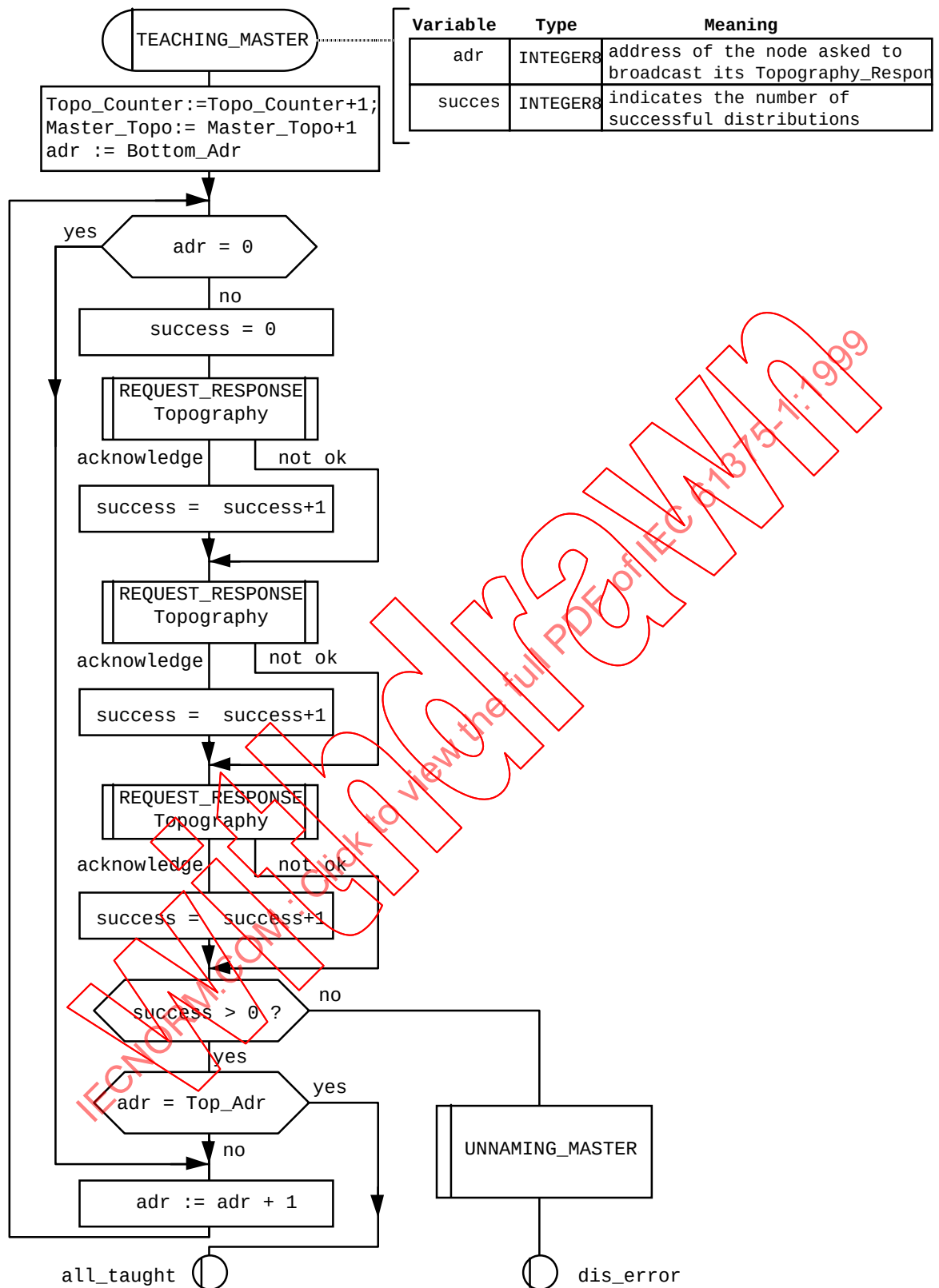


Figure 200 - Macro TEACHING_MASTER

4.7.4.8.8 Macro 'UNNAMING_MASTER'

As shown in Figure 201, the master disbands its composition by unnamings all nodes. To this effect, it shall broadcast three Unname Requests in sequence within 1,0 ms, and then, if it is a strong node go to the macro NAMING_MASTER or else, go to the macro UNNAMED_SLAVE.

NOTE The delay is necessary to ensure that all nodes receive Unname Request, a slave will observe that same delay before going to End Setting.

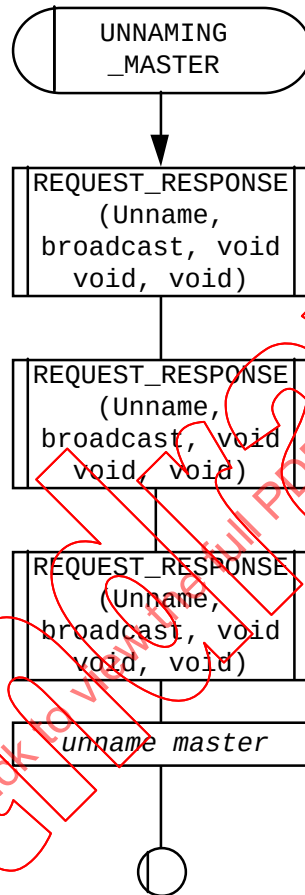


Figure 201 - Macro 'UNNAMING_MASTER'

4.7.4.8.9 Macro 'REGULAR_MASTER'

As shown in Figure 202, the master is in regular operation.

The macro 'REGULAR_MASTER' is divided into three blocks:

- a) 'PERIODIC_POLL': the master polls the nodes in its Periodic_List for Process Data. This phase is detailed in the 4.7.4.8.11;
- b) 'SUPERVISORY_POLL':
 - the master shall send a Presence_Request to one End Node in each basic period; the interval between any four consecutive Presence_Requests in the same direction shall be less than $6.5 \times T_{bp}$. A convenient way to achieve this is to send Presence_Request at a fixed percentage of the Basic Period, for instance at the beginning of each Basic_Period.
 - the master checks the status of the nodes which raised the 'C' bit. This checking shall take place after the Periodic_Phase.

- c) 'MESSAGE_POLL': The master polls the nodes for Message Data in the remaining time before the next Periodic_Phase begins, as specified in 4.7.4.8.12

The master shall leave the macro 'REGULAR_MASTER':

- a) if it ceases to sense the presence of an End Node for three consecutive polls, disband its composition and then go to the macro 'NAMING_MASTER';
- b) if it detects the presence of another nameable composition and that inauguration is enabled, and then execute UNNAMING_MASTER to disband its composition and then go to the macro 'NAMING_MASTER'
- c) if it detects a composition change, and then go to the macro 'TEACHING_MASTER';
- d) if it finds a composition stronger than its own with inauguration enabled and then disband its composition before going to UNNAMED_SLAVE;
- e) if it has been set asleep or has been disconnected by an application command and then go to the macro 'UNNAMED_SLAVE';

NOTE 'UNNAMED_SLAVE' brings the node to the sleep mode when all other nodes are also in this state.

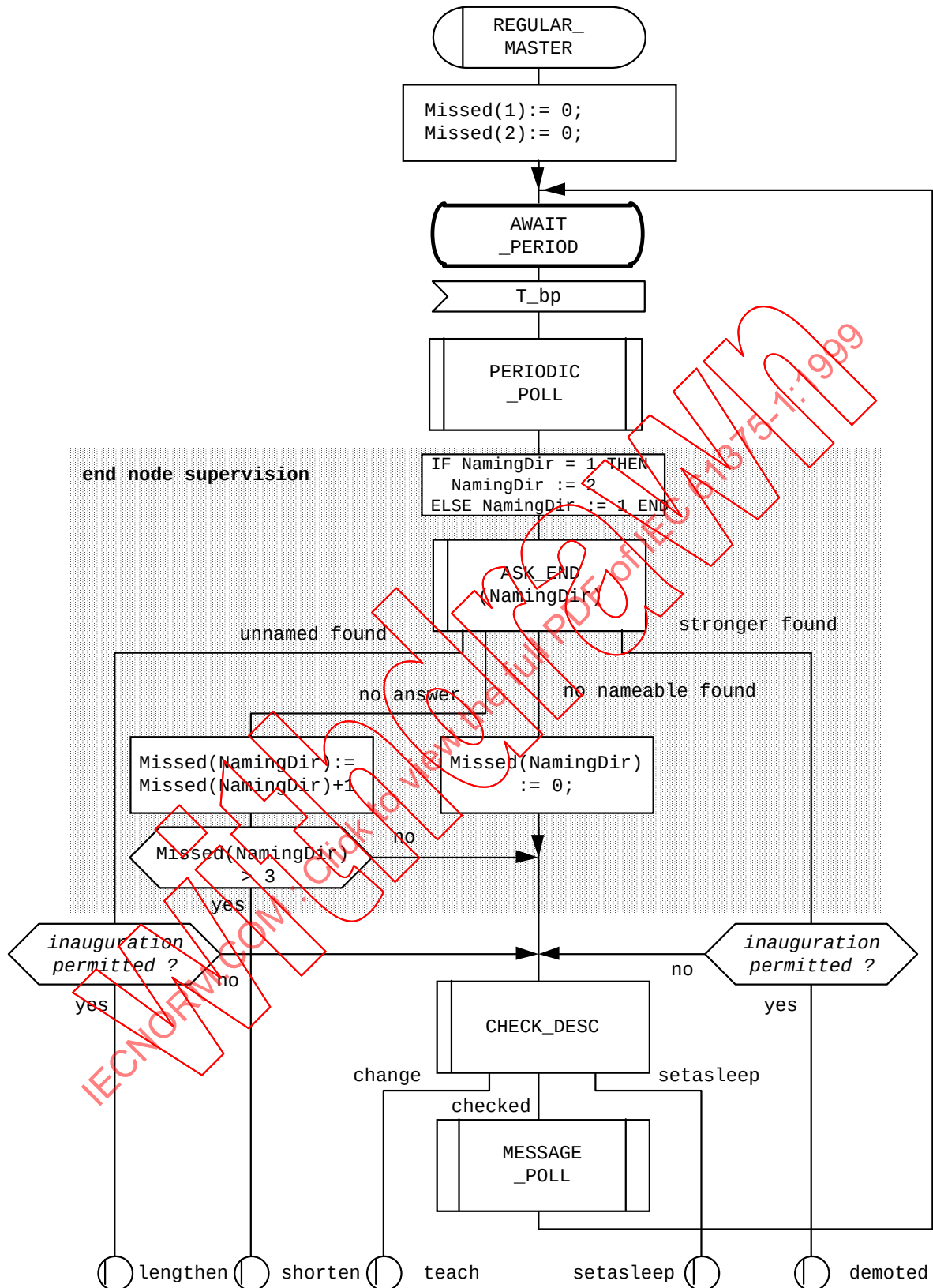


Figure 202 - Macro 'REGULAR_MASTER'

4.7.4.8.10 Macro 'CHECK_DESC'

As shown in Figure 203, this macro checks for any Intermediate_Nodes which asked to change descriptor or to be put to sleep.

The case of the End Nodes is treated separately, since an End Node could in addition signal a bus lengthening.

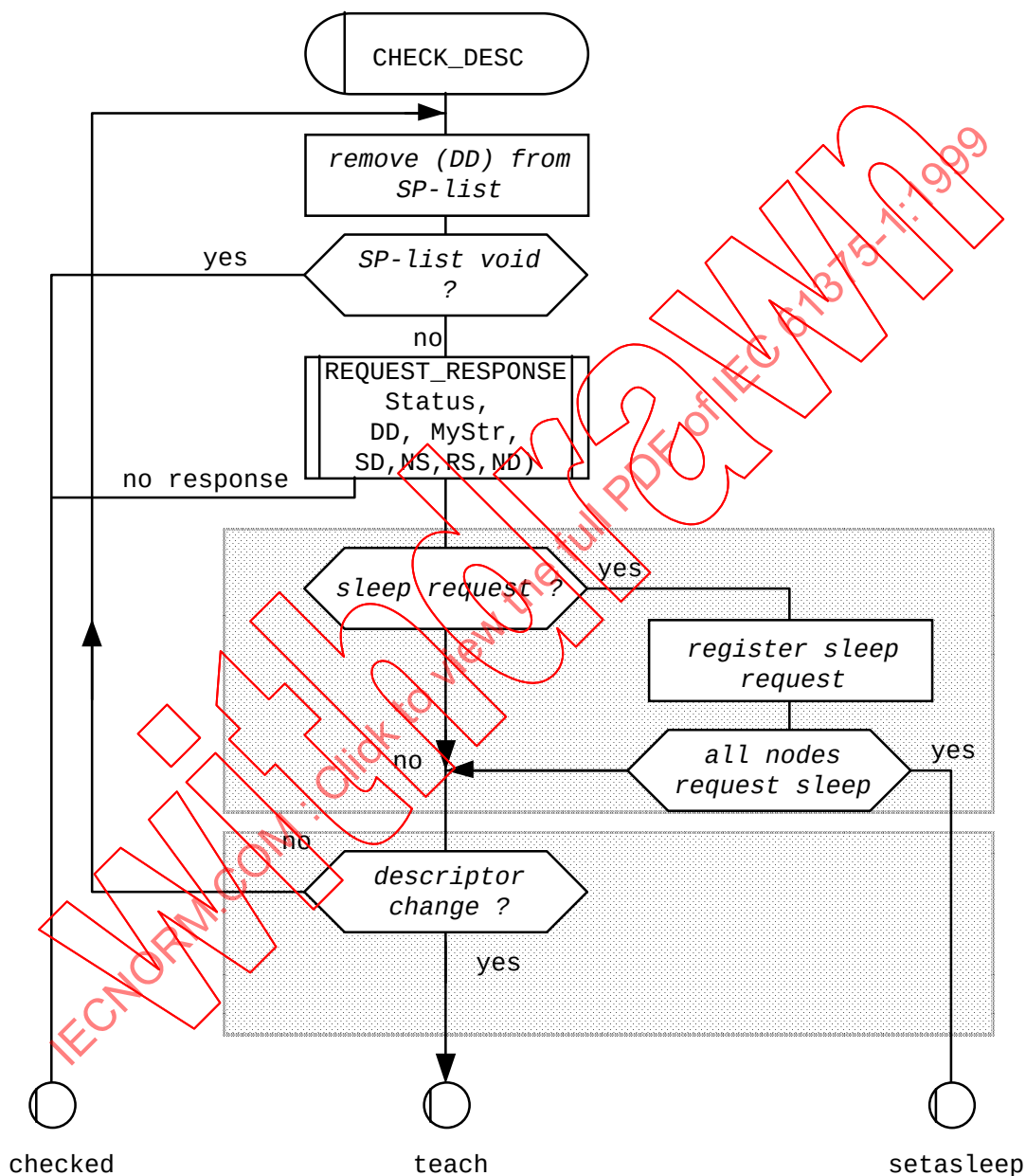


Figure 203 - Macro CHECK_DESC

4.7.4.8.11 Macro 'PERIODIC_POLL'

As shown in Figure 204, the master scans the nodes according to the Periodic_List, which lists the addresses to be polled in this period.

When polling the nodes for Process Data, the master shall:

- a) record the existence and the Process Data of the polled node.
- b) record composition changes (C_bit) from the nodes which changed their Node Descriptor;
- c) record nodes which require Message Data transfer by raising their A_bit;
- d) record nodes which inhibit inauguration by their 'I_bit'.

The master shall not repeat Process_Data_Request in the same Basic Period if it receives no Process_Data_Response.

When the master polls itself, it shall send a Process_Data_Request to itself before sending a Process_Data_Response.

IECNORM.COM: Click to view the full PDF of IEC 61375-1:1999

Withd 2M

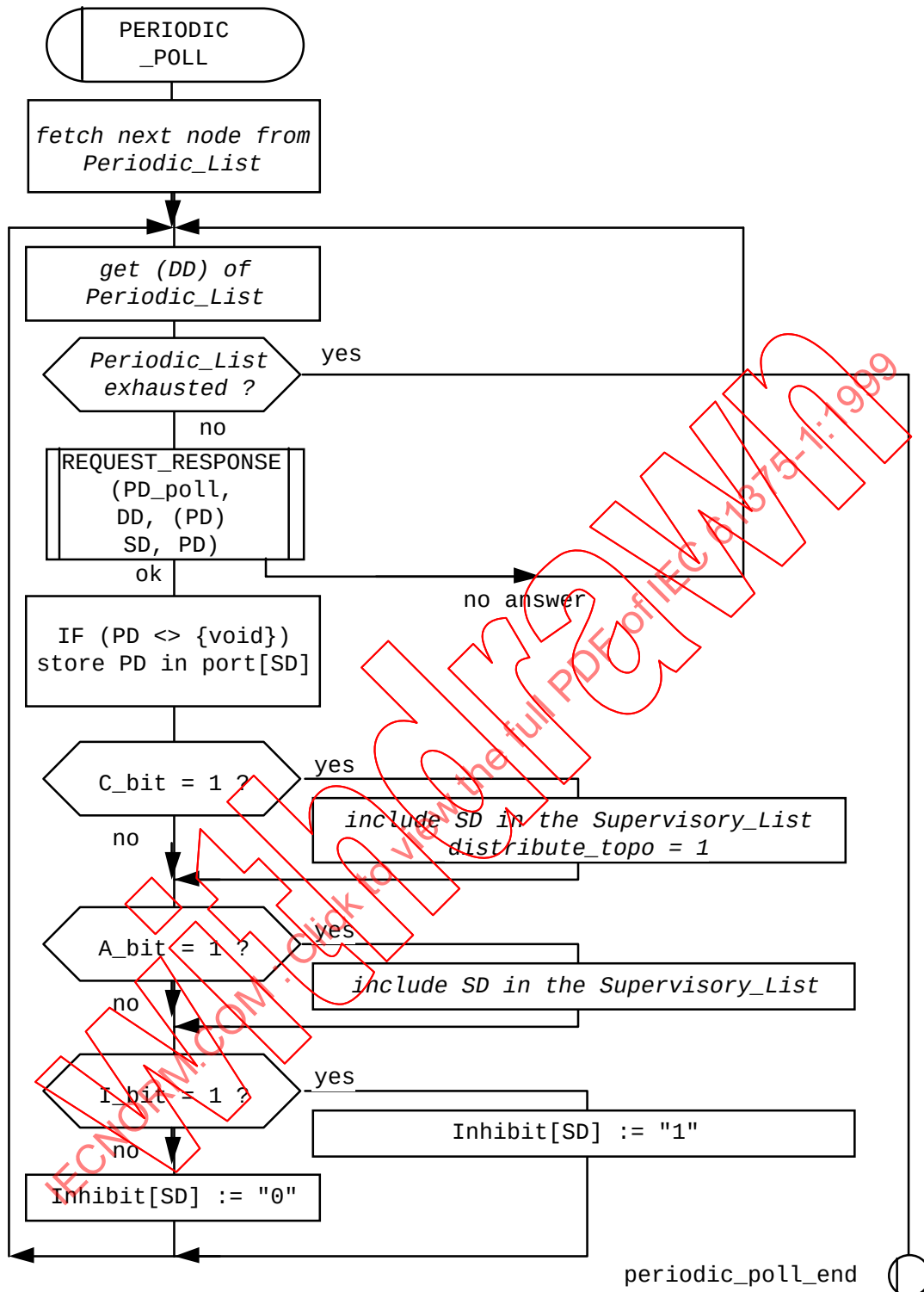


Figure 204 - Macro PERIODIC_POLL

4.7.4.8.12 Macro 'MESSAGE_POLL'

As shown in Figure 205, the master shall send Message Data if there is sufficient time left before the next Periodic_Phase.

The master shall not repeat Message Data Request in the same Basic Period if it receives no Message Data Response.

When the master polls itself, it shall send a Message Data Request to itself before sending a Message Data Response.

The node shall leave that state if there is not more time to send a complete frame before the next periodic phase begins. it shall then return to the state 'AWAIT_PERIOD'.

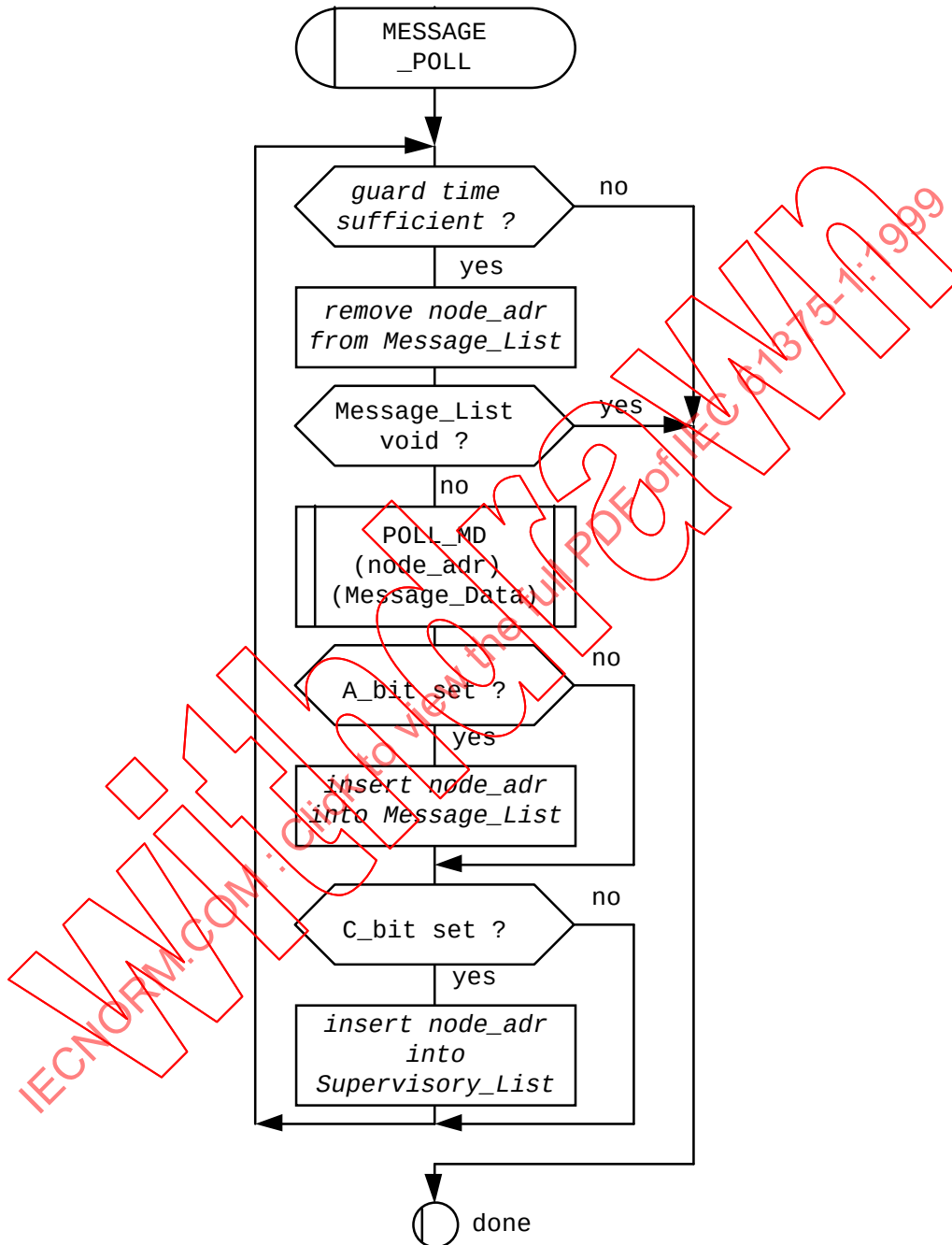


Figure 205 - Macro MESSAGE_POLL

4.7.4.9 Slave States

4.7.4.9.1 Macro 'UNNAMED_SLAVE'

As shown in Figure 206, a node shall put itself in End Setting with both its Auxiliary Channels listening and await naming in the state AWAIT_NAMING.

At entering the AWAIT_NAMING state, a node shall reset the T_await_naming timer and expect:

a) a timer T_await_naming

- if it received a NodeSleep command from the application, switch to Intermediate Setting and go to the low-power state NODE_SLEEP
- if it is configured as a weak node, go to NAMING_MASTER, or
- otherwise return to UNNAMED_SLAVE;

b) a signal 'NamedBy' of the Auxiliary channel,

- locks its Main Channel towards the direction it has been named,
- go to the state NAMED_SLAVE

Withd2Mn
IECNORM.COM: Click to view the full PDF of IEC 61375-1:1999

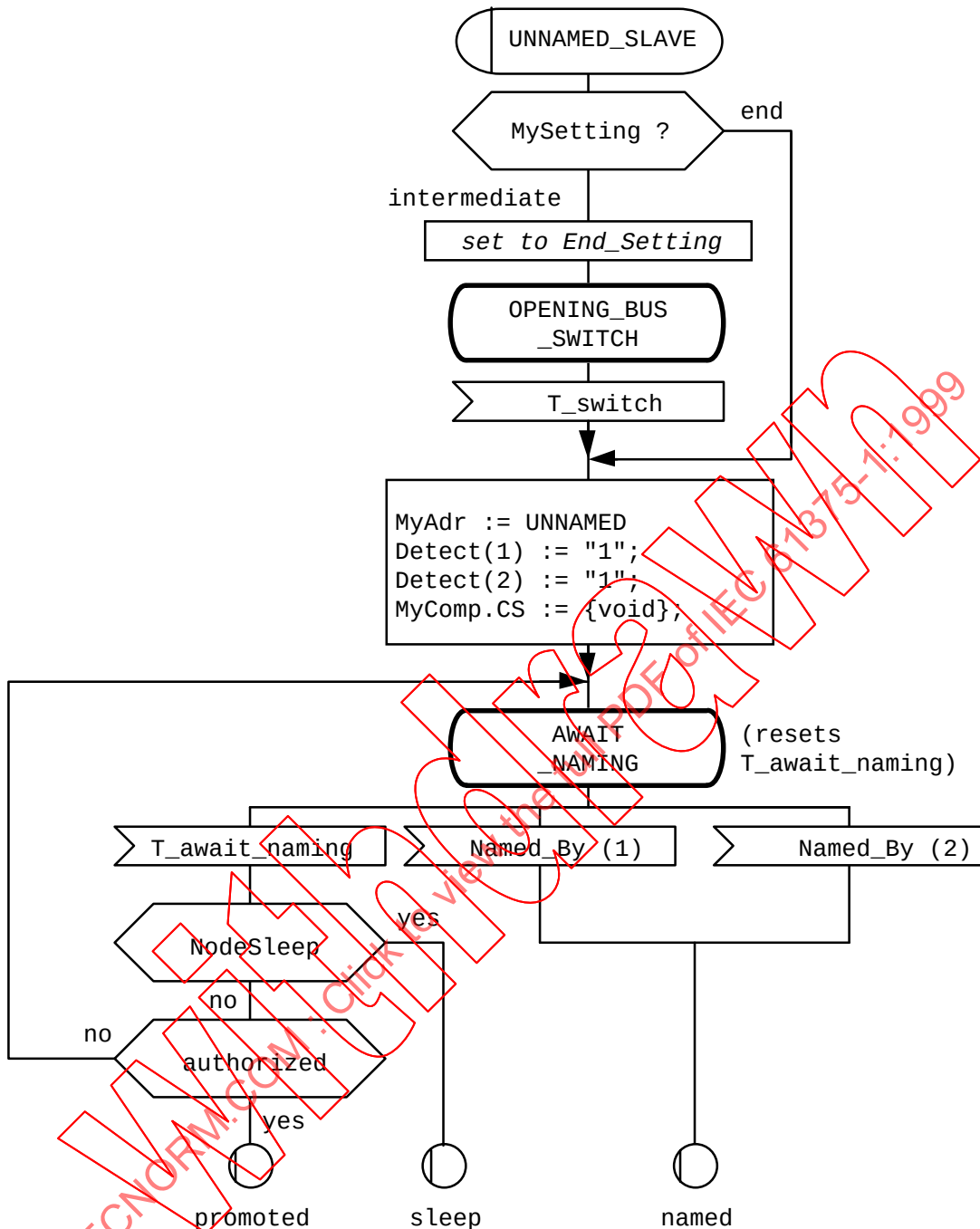


Figure 206 - States 'UNNAMED_SLAVE'

4.7.4.9.2 Macro 'NAMED_SLAVE'

As shown in Figure 207, a node in this state has been named and may change from End Setting to Intermediate Setting or vice-versa within this state in response to the master's requests.

At entering the NAMED_SLAVE state, a node shall reset the T_named_slave timer and expect:

- a) a change in Node Descriptor,
 - if it has been promoted to strong node, go to the NAMING_MASTER macro or
 - otherwise, raise its 'C_bit' and return to NAMED_SLAVE;

- b) a time-out $T_{\text{named\_slave}}$
- go to UNNAMED_SLAVE;
- c) Naming Request:
- send Naming Response to the master;
- d) SetInt Request:
- send SetInt Response to the master, go to Intermediate Setting if it is not already in that setting;
- e) SetEnd Request:
- send SetEnd Response to the master, go to End Setting if it is not already in that setting;
- f) Status Request:
- broadcast Status Response;
- g) Status Response:
- record the status to check line redundancy;
- h) Unname Request:
- wait $T_{\text{aux\_main}}$ (to allow all nodes to receive one of the three Unname Requests),
 - set the threshold for the $T_{\text{await\_naming}}$ timer at the highest value and
 - return to the state 'UNNAMED_SLAVE';
- i) Topography Request, Topography Response:
- respond like in LEARNING_SLAVE and go to LEARNING_SLAVE;
- j) none of the above:
- increment the error counter.

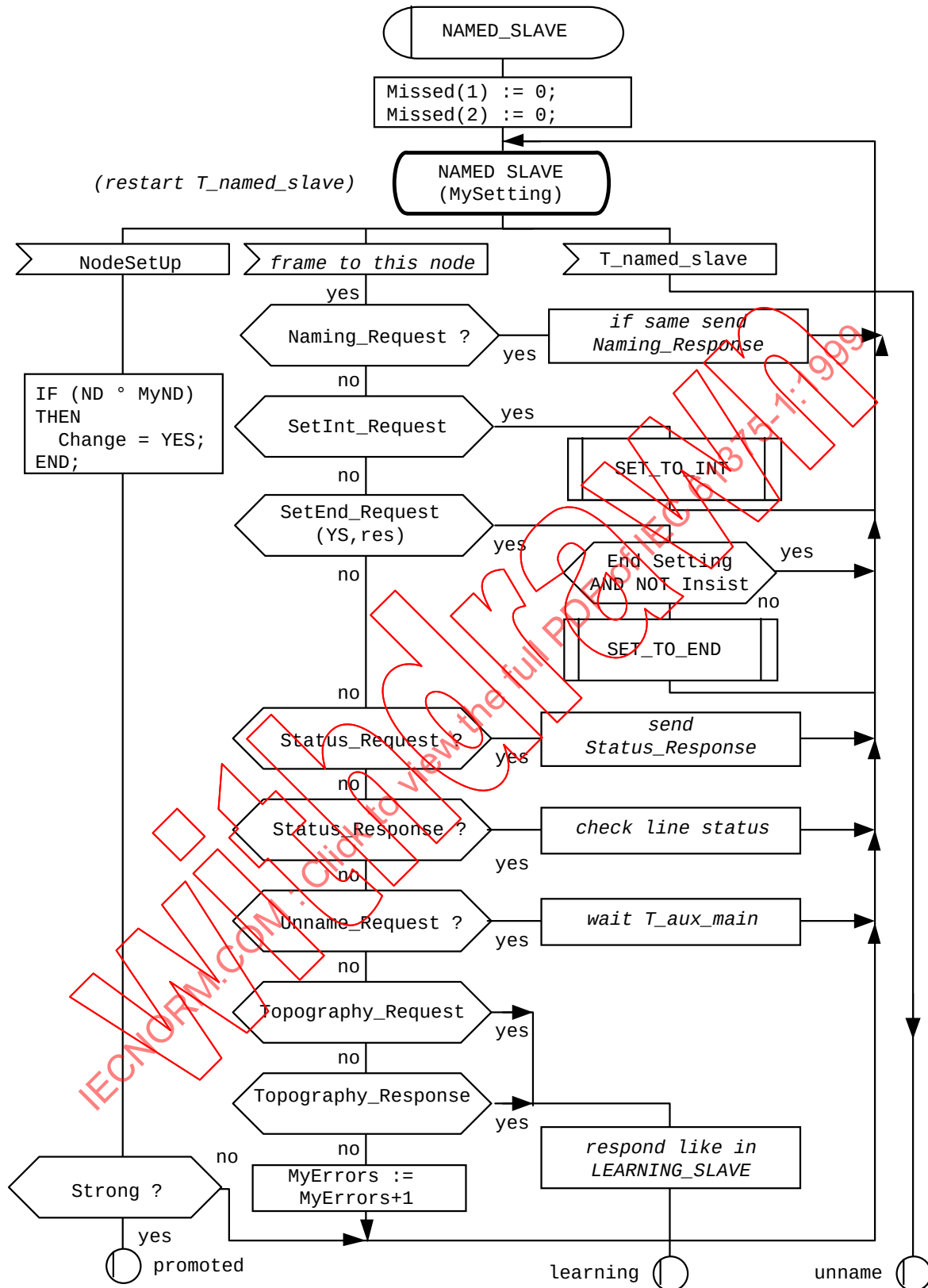


Figure 207 - States 'NAMED_SLAVE'

NOTE 1 In the naming phase, bus supervision is done with the Status Responses, disregarding End Nodes.

NOTE 2 A node ignores Message Data Requests, Message Data Response, Process_Data_Requests and Process_Data_Responses, Presence Requests and Presence Responses in this state.

4.7.4.9.3 Macro 'LEARNING_SLAVE'

As shown in Figure 208, the node receives the Topography information from all other nodes while supervising the bus. The node changes neither its setting nor its name.

At entering the LEARNING_SLAVE state, a node shall reset the T_learning_slave timer and expect:

- a) a change in Node Descriptor
 - if it has been promoted to strong node, go to the NAMING_MASTER state or
 - otherwise, it shall raise its 'C_bit' ;
- b) a time-out T_learning_slave
 - go to UNNAMED_SLAVE without changing its T_await_naming timer value;
- c) Unname Request:
 - wait T_aux_main (to allow all nodes to receive one of the three Unname Requests),
 - set the threshold for the T_await_naming timer at the highest value T_await_max and
 - go to the state 'UNNAMED_SLAVE';
- d) Status Request:
 - send Status Response to the master;
- e) Status Response:
 - register the status;
- f) Topography Request:
 - broadcast its Topography Response;
- g) Topography Response:
 - update its Topography, check if it is in possession of a complete Topography , all Topography_Requests having been received with the same Master_Topo, and if yes, set Updated:= TRUE;
- h) Presence Request, Presence Response, Process_Data_Request, Process_Data_Response, Message_Data_Request, Message_Data_Response
 - respond as if in REGULAR_SLAVE and go to REGULAR_SLAVE.
- i) none of the above:
 - increment the error counter.

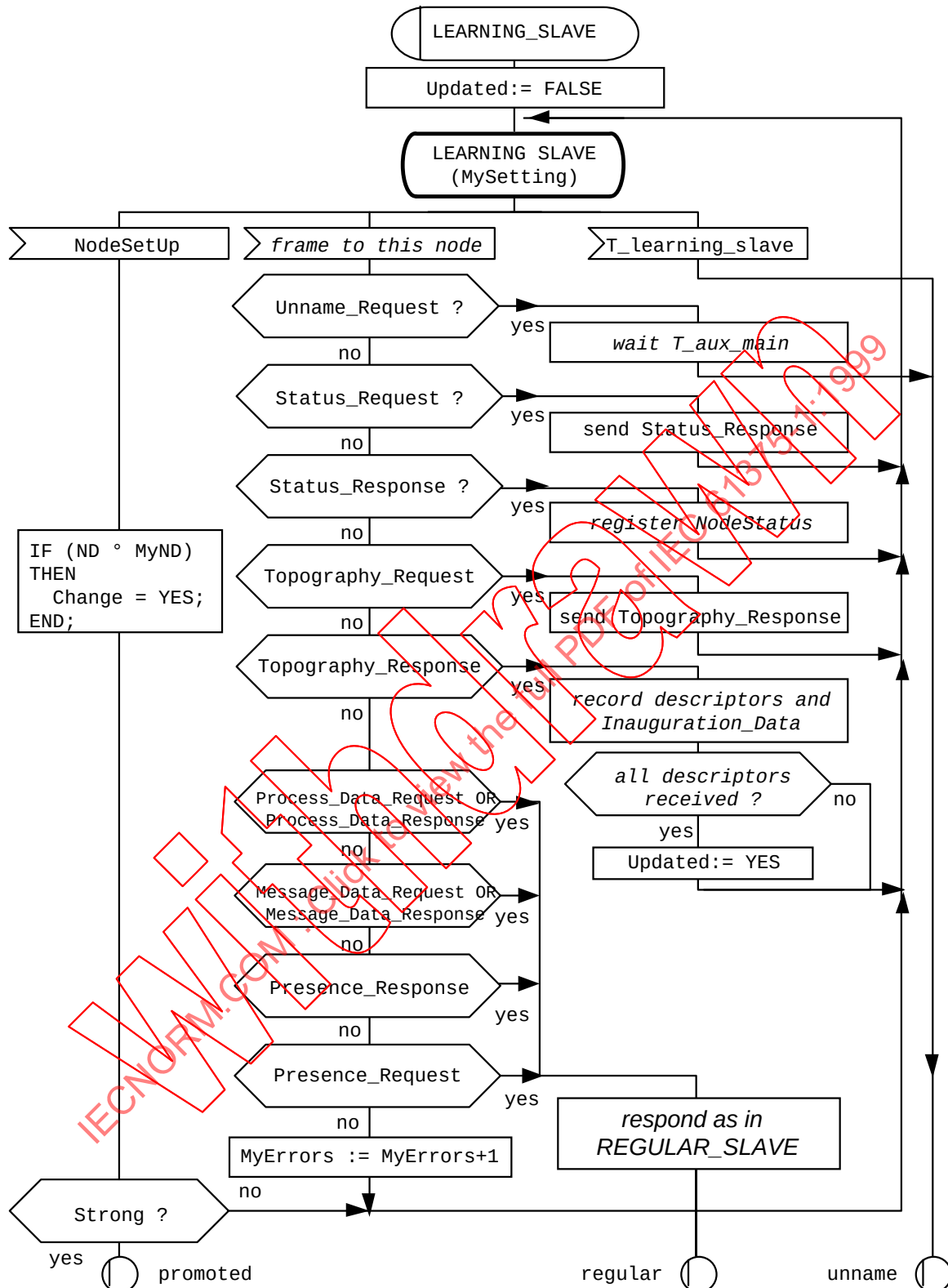


Figure 208 - Macro 'LEARNING_SLAVE'

NOTE In the learning phase, bus supervision is done with the Topography Responses, disregarding End Nodes.

4.7.4.9.4 Macro 'REGULAR_SLAVE'

As Figure 209 shows, this is the regular operational state of a node, in which it sends and receives Process Data and Message Data and signals events by setting the indication bits in its responses. A node supervises the activity of the End Nodes through two timers.

In the REGULAR_SLAVE state, the node shall expect:

- a) a change in Node Descriptor
 - if it has been promoted to strong node go to the NAMING_MASTER state or
 - otherwise, raise its C-bit;
- b) a time-out T_bus_check for each one of the End Nodes it supervises.
 - go to UNNAMED_SLAVE keeping the start value of T_await_naming as it is.
- c) Presence Request:
 - broadcast Presence Response (the node shall ignore Presence Request if it is not an End Node).
- d) Presence Response:
 - restart the corresponding T_bus_check timer.
- e) Status Request:
 - send Status Response to the master;
- f) Unname Request:
 - wait T_aux_main (to allow all nodes to receive one of the three Unname Requests),
 - set the threshold of the T_await_naming timer at the highest value and
 - go to UNNAMED_SLAVE,
- g) Process_Data_Request without Process Data:
 - if the node is Updated, send a Process_Data_Response, reading the data from its source port,
 - otherwise send a void Process_Data_Response.
- h) (optionally) Process_Data_Request with Process Data:
 - if the Updated is TRUE, write these data into the sink port dedicated to direct master data and send a Process_Data_Response, reading the data from its source port,
 - otherwise ignore them and send a void Process_Data_Response;
- i) Process_Data_Response:
 - if Updated is TRUE, write these data in the sink port corresponding to the source address,
 - otherwise ignore them;

j) Message Data Request:

- if the send queue is void, send a void Message Data Response (link_data_size = 0) addressed to the master, or otherwise
- send a Message Data Response with a Message Data packet extracted from its send queue;

k) Message Data Response:

- store the incoming Message Data in its Receive_Queue if there is room, otherwise ignore them (see 4.8.3)

l) Topography Request or Topography_Response:

- set Updated FALSE, respond as in LEARNING_SLAVE and go to that state,

m) none of the above:

- increment the error counter and return to LEARNING_SLAVE.

NOTE 1 The T_bus_check timer may be implemented also with a single timer and counters.

NOTE 2 There is no timer associated with REGULAR_SLAVE and restarted each time this state is entered, since this function is executed by T_bus_check.

NOTE 3 The supervision of the End Nodes can also be done with a T_bus_check timer individually for each end node.

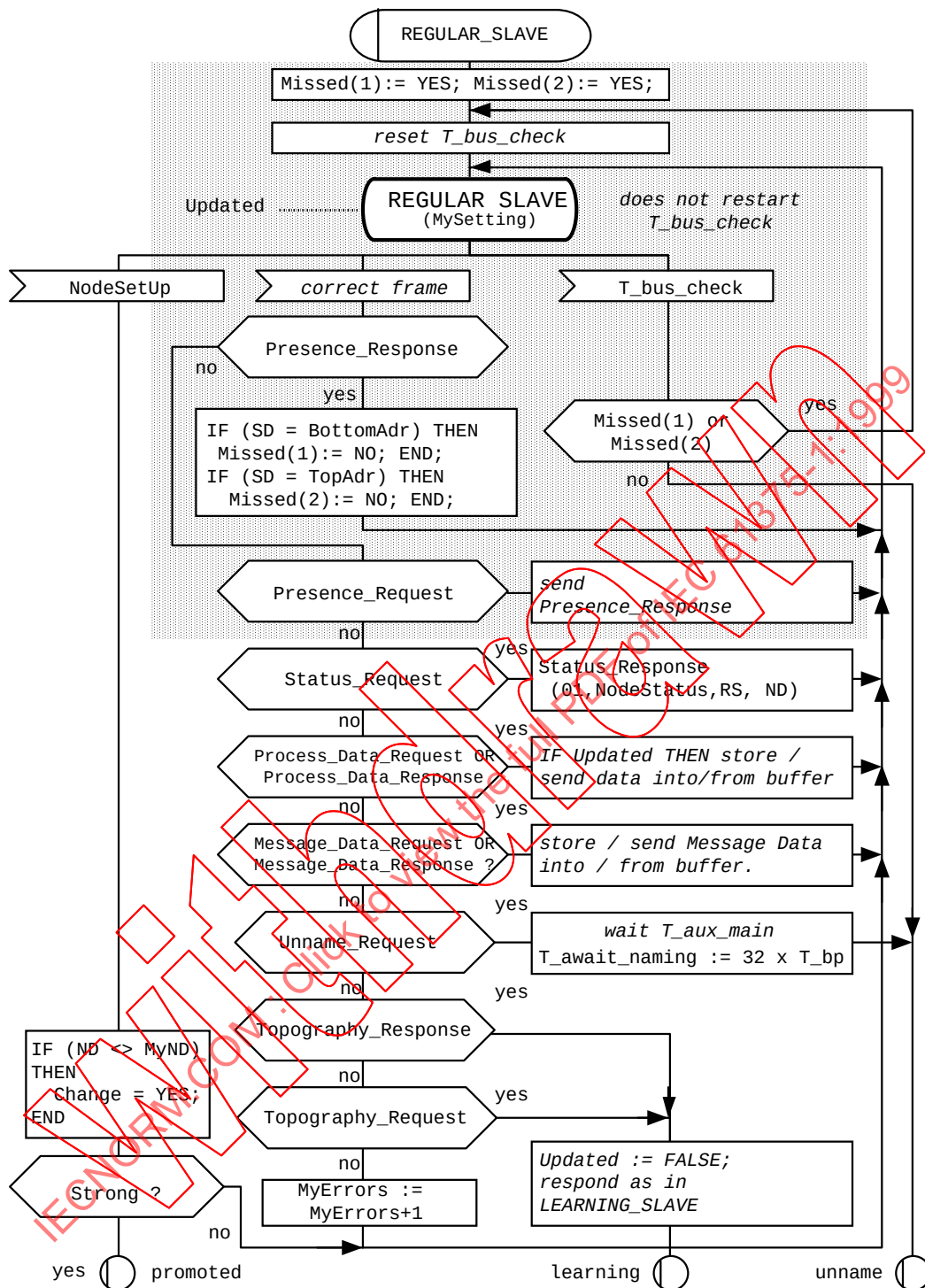


Figure 209 - Macro 'REGULAR_SLAVE'

4.7.4.10 Time-outs

The recommended time-out values ($\pm 20\%$) are listed in Table 71.

Table 71 - Time constant values

Time constant name	Value	Usage
T_await_naming	1) T_await_min = 1,0 ms + T_switch	Master renaming its composition
	2) $((63 - \text{MyAdr}) + 0.5) \times T_{bp}$	Nodes in direction 1 of the master
	3) $(\text{MyAdr}-1) \times T_{bp}$	Nodes in direction 2 of the master
	4) T_await_max = 32 x T_bp	Initialised or explicitly unnamed nodes.
T_aux_main	1,0 ms	Switch delay from auxiliary to main channel (or the reverse)
T_await_response	1,756 ms	Time the master waits for a Slave Frame. This time accounts for the longest possible frame since HDLC controllers can only signal the end of a frame.
T_bp	25,0 ms	Basic Period (regular operation)
T_naming_master	15,0 ms	naming period (inauguration)
T_named_slave	15,0 ms	master supervision during naming phase
T_learning_slave	15,0 ms	master supervision during learning phase
T_bus_check (1) T_bus_check(2)	6,5 x T_bp	Supervision of the end nodes during regular operation
T_detecting	2 x T_naming_master	Interval between Detect_Requests (if any) during inauguration.
	2 x T_bp	Interval between Presence Requests / Detect_Requests (if any) during regular operation.
T_detecting_response	1,047 ms	Time the End Node waits for a Detect Response.
T_new_inaug	n x T_bp	Minimum time between two consecutive inaugurations. n is set by the application
MAXLOST	50	T_detecting x MAXLOST is the time after which an End Node assumes that the other composition is no longer present
T_switch	10,0 ms	Default delay to close or open the relays.

4.8 Link layer interface

4.8.1 Link layer layering

The Link Layer Interface provides three services, as shown in Figure 210:

- The Link Process_Data_Interface (LPI), which is used by the Variables Service is specified in Clause 2 (Real-Time Protocols). Only the parameters specific to the WTB are specified in this standard;
- The Link Message_Data_Interface (LMI), which is used by the Messages Service is specified in Clause 2 (Real-Time Protocols). Only the parameters specific to the WTB are specified in this standard;
- The Link Supervision_Interface (LSI), which allows the configuration of the link layer and the supervision of the bus is specific to the WTB and specified in this standard.

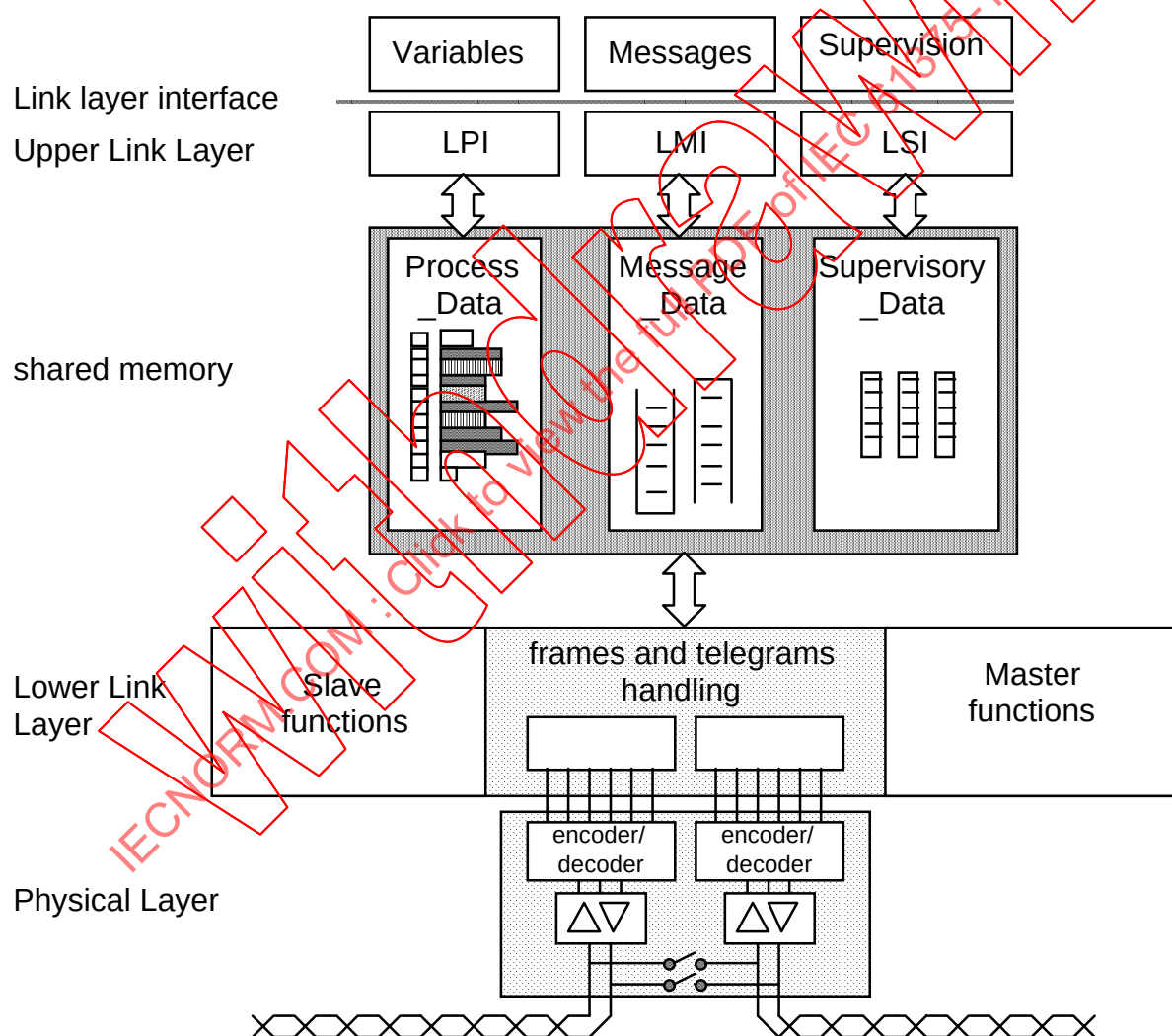


Figure 210 - Link layer layering

4.8.2 Link Process_Data_Interface

4.8.2.1 General

The interface for Process Data between the link layer and the higher layers is a shared memory called the Traffic_Store, which can be simultaneously accessed by the bus and by the application.

The Traffic_Store is structured as a number of Ports, which contain exactly one frame ready for transmission or reception of Process Data.

Each Port is identified within a node by a Traffic_Store identifier and a 12-bit Port address.

The implementation of the Ports is not part of standardisation.

Clause 2 'Real-Time Protocols' specifies the Port access.

4.8.2.2 WTB specific

The Traffic_Store of the WTB shall hold up to 64 Ports, each with a maximum of 1024 bits.

In all cases:

- each node shall have one source port to broadcast its Process Data, the most significant six bits being '000000'B and the least significant six bits of the port address being the address of this node;
- each node shall have a sink port to receive Process Data from each other possible node on the bus, the most significant six bits being '000000'B and the least significant six bits of the port address being the address of the source node.

In applications where the master includes Process Data in its Process_Data_Request, intended for the polled slave only.

- the master shall have one source port to send Process Data to each possible slave, the most significant bits being '000010'B and the least significant six bits of the port address being the destination node address;
- each slave shall implement one sink port to receive the master's Process Data, the most significant bits being '000010'B and the least significant six bits of the port address being its node address.

A node shall not accept Process Data from a node that was not included in the received Topography or that it cannot interpret the descriptor received in the Topography.

A node may accept Process Data from another node whose Node_Type it knows, but whose Node_Version is different from the one it knows, but it shall decode the data according to the lower of both Node_Versions.

A node may not change the format of its Process_Data_Response without receiving a Topography containing its new Node_Key.

It is recommended to reserve the first two octets of Process Data to identify the type of frame (Node_Type + Node_Version = Node_Key) as a further protection.

4.8.3 Link Message_Data_Interface

4.8.3.1 General

The Link Message_Data_Interface (LMI) specified in Clause 2 'Real-Time Protocols' provides services for sending Message Data frames and services for retrieving received Message Data frames. In addition, send confirmation and receive indication services are supported.

The Link Message_Data_Interface provides the basic service on which all higher protocols are built:

- a) The network layer provides routing through the network and directory functions;
- b) The transport protocol provides a half-duplex end-to-end control of messages;
- c) The session layer pairs messages to provide a Remote Procedure Call;
- d) The presentation layer unifies the data presentation;
- e) The application interface provides the client and server interfaces

4.8.3.2 Packet size

The 'link_data_size' field in a void packet shall be zero.

The 'link_data_size' field shall not be larger than 128.

4.8.3.3 Protocol_Type

The Protocol_Type for the Real-Time Protocols shall be indicated by the link_control field set to '00xxx111'B (Message Data Response).

4.8.3.4 Message transport protocol

A WTB node shall not announce a packet size larger than 124 octets in its Connect_Request.

When responding to a Connect_Request, a node shall specify in its Connect_Response a packet size equal to 124 octets, or the proposed packet size, whichever is smaller.

4.8.4 Link management interface

4.8.4.1 General

The link management interface is specific to the WTB.

It provides general services for configuration and inspection of the link layer and event reporting.

The following Subclauses do not imply a particular implementation. Any interface which provides the same semantics is allowed.

The format of the parameters of the following interface procedures is not prescribed. However, the Train Network Management standard (Clause 5) proposes a message format which is recommended as a parameter format.

4.8.4.2 Interface procedures

The procedures of this interface are prefixed by ls_t (link supervision, WTB).

4.8.4.2.1 Type LS_T_RESULT

The procedure returns are of the type LS_T_RESULT:

Constant	Code	Meaning
L_OK	0	success
L_BUSY	1	try again later
L_CALLING_SEQUENCE	2	wrong command sequence
L_MISSING_UDF	3	user-defined function unknown
L_CONFIGURATION_INVALID	4	Topography or node list invalid

4.8.4.2.2 Constants LS_T_STATE

The following constants indicate in which major state the node is currently.

Constant	Code	Meaning
LS_INITIALIZED	0	node in state UNCONFIGURED
LS_CONFIGURED	1	node in state CONFIGURED
LS_READY_TO_NAME	2	node in state NAMING_MASTER
LS_READY_TO_BE_NAMED	3	node in state UNNAMED_SLAVE
LS_INHIBITED	4	node Inauguration inhibited
LS_REGULAR_STRONG	5	node in state REGULAR_MASTER (strong) or
LS_REGULAR_SLAVE	6	TEACHING_MASTER
LS_REGULAR_WEAK	7	node in state REGULAR_SLAVE or TEACHING_MASTER
		node in state REGULAR_MASTER (weak) or
		TEACHING_MASTER

NOTE A node cannot indicate that it is in sleep state.

4.8.4.3 Reporting

4.8.4.3.1 Procedure ls_t_Report

Action	Reports to the user a change in the link layer. This procedure is called by the link layer and shall be previously subscribed (see: ls_t_Configure).	
Syntax	Typedef LS_T_RESULT (* ls_t_Report) (ls_report)	
Input	ls_report	one of the LR_REPORT report codes.

4.8.4.3.2 Constants LR_REPORT

The value of the report codes shall be as follows:

Constant	Code	Meaning
LR_CONFIGURED	16	link layer is configured
LR_STRONG	17	node is now operational master
LR_SLAVE	18	node is now operational slave
LR_PROMOTED	19	node changed from weak to strong master
LR_NAMING_SUCCESSFUL	20	master indicates end of inauguration
LR_NAMED	21	node is a named slave
LR_WEAK	22	master changed to weak master
LR_REMOVED	23	node removed from configuration
LR_DEMOTED	24	weak master detected a strong master
LR_DISCONNECTION	25	node disconnected
LR_INHIBITED	26	inauguration inhibited
LR_INCLUDED	27	included in composition
LR_LENGTHENING	28	master detected train lengthening
LR_DISRUPTION	29	node detected loss of End Node
LR_MASTER_CONFLICT	30	strong master detected another strong master
LR_NAMING_FAILED	31	failure while naming
LR_NEW_TOPOGRAPHY	32	reception of a new Topography
LR_NODE_STATUS	33	naming started
LR_POLL_LIST_OVF	34	partially operational
LR_ALLOWED	35	missing node

4.8.4.4 Initialisation service

4.8.4.4.1 Procedure ls_t_Init

Action	Initialises the link layer and sets variables to predefined values. After the call, the link layer shall be ready to receive commands. This procedure shall be called only once after a hardware reset. This procedure is implementation-dependent.
Syntax	LS_T_RESULT ls_t_Init (void);

4.8.4.5 Reset service

4.8.4.5.1 Procedure ls_t_Reset

Action	Resets the link layer to predefined values. After the call, the link layer shall be in the idle state, ready to receive commands.
Syntax	LS_T_RESULT ls_t_Reset (void);

4.8.4.6 Configuration service

4.8.4.6.1 Procedure ls_t_Configure

Action	Configures the link layer. After the call, the node shall be ready to start a communication	
Syntax	<pre> LS_T_RESULT ls_t_Configure (Type_Configuration * p_configuration); </pre>	
Input	p_configuration	Pointer to the following configuration data structure.

4.8.4.6.2 Type_NodeKey

A data structure of Type_NodeKey shall contain the following elements:

Attribute	Type	Meaning
node_type	UNSIGNED8	type of the node as indicated by the application
node_version	UNSIGNED8	version of the node as indicated by the application

NOTE Type_NodeKey is the C-Type corresponding to the structure Node_Key (see 4.7.2.1).

4.8.4.6.3 Type_NodeDescriptor

A data structure of Type_NodeDescriptor shall contain the following elements:

Attribute	Type	Meaning
node_frame_size	UNSIGNED8	size of the Process Data frame in octets.
node_period	UNSIGNED8	value of the Node_Period in 2n multiples of the Basic Period (node_period is the value of n).
node_key	Type_NodeKey	see this data type.

NOTE Type_NodeDescriptor is the C-Type corresponding to the structure Node_Descriptor (see 4.7.2.1).

4.8.4.6.4 Type_Configuration

The data structure Configuration shall contain the following elements:

Attribute	Type	Meaning
transmission_rate	UNSIGNED16	transmission rate in kbit/s, default: 1000 kbit/s
basic_period	UNSIGNED16	Basic Period in milliseconds, default: 25,0 ms
fritting_disabled	UNSIGNED16	= 1 if fritting disabled, default: 0
node_descriptor	Type_NodeDescriptor	see 4.8.4.6.3
poll_md_when_idle	UNSIGNED8	=1 if background scanning enabled, default: 0 (See 4.6.3.4).
sink_port_count	UNSIGNED16	Maximum number of sink ports, default: 22
source_port_count	UNSIGNED16	Maximum number of source ports, default: 1
port_size	UNSIGNED8	Maximum length of a Port in octets, default: 128
p_traffic_store	WORD32	Pointer to the traffic store, default: NULL
ls_t_report	WORD32	Call back function for reports, default: NULL
max_number_nodes	UNSIGNED8	Maximum number of nodes whose inauguration data to be stored, default: 0
inaug_data_max_size	UNSIGNED8 (≤ 124)	Maximum number of octets of application defined inauguration data to send, default: 0
s_inaug_data_size	UNSIGNED8 (≤ 124)	Actual number of octets of application defined inauguration data to send, default: 0
p_inaug_data_list	WORD32	Pointer to data area where to copy inauguration data from, default: NULL

4.8.4.6.5 Type_Inauguration_Data

The data structure Type_Inauguration_Data shall contain the following elements:

Attribute	Type	Meaning
inaug_data_max_size	UNSIGNED8 (≤ 124)	maximum number of octets of application defined inauguration data stored, default: 0
nr_descriptors	UNSIGNED8	number of nodes whose inauguration data is stored, default: 0 = invalid
node_descriptions	ARRAY [nr_descriptors] OF	list of the application defined inauguration data for each WTB node, consisting of:
node_type	WORD8	first part of Node_Key
node_version	WORD8	second part of Node_Key
sam	BOOLEAN1	'1' if same orientation as master
rsv1	WORD1 (=0)	reserved, =0
node_address	UNSIGNED6	address of node from which inauguration data was received
inauguration_data_size	UNSIGNED8	size of Inauguration_Data (≤ 124 octets)
inauguration_data	ARRAY[inaug_data_len] OF WORD8	application defined inauguration data

NOTE 1 The 'node_descriptions' shall be initialised before inauguration starts.

NOTE 2 At the end of the inauguration, the table 'node_descriptions' contains 'nr_descriptors' rows, one for each node.

4.8.4.7 Set Slave service

4.8.4.7.1 Procedure ls_t_SetSlave

Action	Prevents a node from becoming master.
Syntax	LS_T_RESULT ls_t_SetSlave (void);

4.8.4.8 Set Weak service

4.8.4.8.1 Procedure ls_t_SetWeak

Action	Enables a node to become weak master.
Syntax	LS_T_RESULT ls_t_SetWeak (void);

4.8.4.9 Set Strong service

4.8.4.9.1 Procedure ls_t_SetStrong

Action	Commands a node to become strong master.
Syntax	LS_T_RESULT ls_t_SetStrong (void);

NOTE This command causes an inauguration.

4.8.4.10 StartNaming service

4.8.4.10.1 Procedure ls_t_StartNaming

Action	Commands the node to start an inauguration.
Syntax	LS_T_RESULT ls_t_StartNaming (void);

4.8.4.11 Remove service

4.8.4.11.1 Procedure ls_t_Remove

Action	Commands the node to remove itself from the configuration and go to a passive state.
Syntax	LS_T_RESULT ls_t_Remove (void);

4.8.4.12 Inhibit service

4.8.4.12.1 Procedure ls_t_Inhibit

Action	Prevents a bus lengthening if additional nodes are detected.
Syntax	LS_T_RESULT ls_t_Inhibit (void);

4.8.4.13 Allow service

4.8.4.13.1 Procedure ls_t-Allow

Action	Enables a bus lengthening if additional nodes are detected.
Syntax	LS_T_RESULT ls_t-Allow (void);

4.8.4.14 SetSleep service

4.8.4.14.1 Procedure ls_t_SetSleep

Action	Causes the node to signal a sleep request.
Syntax	LS_T_RESULT ls_t_SetSleep (void);

4.8.4.15 CancelSleep service

4.8.4.15.1 Procedure ls_t_CancelSleep

Action	Causes the node to cancel a sleep request.
Syntax	LS_T_RESULT ls_t_CancelSleep (void);

4.8.4.16 GetStatus service

4.8.4.16.1 Procedure ls_t_GetStatus

Action	Retrieves the status of the physical and of the link layer.	
Syntax	LS_T_RESULT Type_WTBStatus*	ls_t_GetStatus (p_status);
Input	p_status	pointer to the place where to put the WTB_Status data structure.

4.8.4.16.2 Type Node_Status

Attribute	Type	Meaning
node_report	BITSET8	'C' declaration corresponding to Node_Report (4.7.2.2.)
user_report	BITSET8	'C' declaration corresponding to User_Report (4.7.2.3)

4.8.4.16.3 Type_WTBStatus

Attribute	Type	Meaning
wtb_hardware_id	UNSIGNED8	identification of the hardware
wtb_software_id	UNSIGNED8	identification of the version of the link layer software.
hardware_state	ENUM8	0: LS_OK correct operation 1: LS_FAIL, hardware failure
link_layer_state	LS_T_STATE	see type definition
net_inhibit	ENUM8	1: some node inhibits inauguration
node_address	UNSIGNED8	node address as assigned by inauguration
node_orient	UNSIGNED8	orientation of the node relative to the master: 0: L_UNKNOWN 1: L_SAME 2: L_INVERSE
node_strength	UNSIGNED8	strength of the node 0: L_UNDEFINED 1: L_SLAVE 2: L_STRONG 3: L_WEAK
node_descriptor	Type_NodeDescriptor	see type definition
node_status	Type_Node_Status	see type definition

4.8.4.17 Get WTB nodes service

4.8.4.17.1 Type_NodeList

A data structure of Type_NodeList shall contain the following elements:

Attribute	Type	Meaning
nr_nodes	UNSIGNED8	number of nodes in the composition.
bottom_node	UNSIGNED8	address of End Node in Direction_1 from the master. The two most significant bits of this octet are 0.
top_node	UNSIGNED8	address of End Node in Direction_2 from the master. The two most significant bits of this octet are 0
node_status_list	ARRAY [MAX_NODES] OF	a list of Node Status, beginning with the bottom node, in the order in which the nodes are located, and ending with the top node, consisting of:
node_status	Type_Node_Status	see type definition

4.8.4.17.2 Procedure ls_t_GetWTBNodes

Action	Reads the list of Node Report and User Report of all nodes in the Topography.	
Syntax	<pre> LS_T_RESULT ls_t_GetWTBNodes (Type_NodeList * p_nodes); </pre>	
Input	p_nodes	pointer to location where to put the list of nodes.

4.8.4.18 Get Topography service

4.8.4.18.1 Procedure ls_t_GetTopography

Action	Allows the application to read the Topography distributed before regular operation begins.	
Syntax	<pre> LS_T_RESULT ls_t_GetTopography (Type_Topography * p_topography); </pre>	
Input	p_topography	pointer to the place where to put the Topography.
Result		returns L_CONFIGURATION_INVALID if the Topography is not valid.

4.8.4.18.2 Type_Topography

The data structure Topography shall contain the following elements:

Attribute	Type	Meaning
node_address	UNSIGNED8	The two most significant bits of this octet are 0. The lower-order 6-bits of this octet are the address of the node to which this station is connected
node_orient	UNSIGNED8	Orientation of the node relative to the master : 0: L_UNKNOWN 1: L_SAME 2: L_INVERSE
topo_counter	UNSIGNED8	The least significant six bits copy the six bits of the Topo Counter of the node, the two most significant bits of this octet are 0.
individual_period	UNSIGNED8	Period allocated to a node by the master as power of two of the basic period in ms.
is_strong	UNSIGNED8	1: bus controlled by a strong master, 0: bus controlled by a weak master
number_of_nodes	UNSIGNED8	number of nodes according to inauguration result
bottom_address	UNSIGNED8	Address of End Node in Direction_1 from the master, the two most significant bits of this octet are 0.
top_address	UNSIGNED8	Address of End Node in Direction_2 from the master, the two most significant bits of this octet are 0.
inauguration_data	Type_Inauguration_Data	see type definition

4.8.4.19 Change Node Descriptor service

4.8.4.19.1 Procedure ls_t_ChgNodeDesc

Action	Provides a new descriptor to the link layer. Calling this procedure during regular operation causes a disruption of traffic and a new distribution of topography.	
Syntax	LS_T_RESULT	ls_t_ChgNodeDesc (Type_NodeDescriptor * node_descriptor);
Input	node_descriptor	pointer to a Node_Descriptor data structure

4.8.4.20 Change User Report service

4.8.4.20.1 Procedure ls_t_ChgUserReport

Action	Allows the application to modify User Report.	
Syntax	<pre> LS_T_RESULT ls_t_ChgUserReport (UNSIGNED8 set_mask UNSIGNED8 clear_mask); </pre>	
Input	set_mask	set the bits set in the mask to 1 in User_Report.
	clear_mask	clears the bits set in the mask to 1 in User_Report.

4.8.4.21 Change Inauguration_Data service

4.8.4.21.1 Procedure ls_t_ChgInauguration_Data

Action	Allows the application to modify this node's Inauguration Data	
Syntax	<pre> LS_T_RESULT ls_t_ChgInauguration_Data (UNSIGNED8 inaug_data_size void* p_inauguration); </pre>	
Input	inaug_data_size	size in octets of the inauguration data (<= 124)
	p_inauguration	user-defined inauguration data

4.8.4.22 Get Statistics service

4.8.4.22.1 Procedure ls_t_GetStatistics

Action	Provides statistical information about usage and errors	
Syntax	<pre> LS_T_RESULT ls_t_GetStatistics (Type_LLStatisticData * p_statistic_data); </pre>	
Input	p_statistic_data	pointer to the statistic data structure (see 4.8.4.22.3)

4.8.4.22.2 Type_LineStatus

A data structure of Type_LineStatus shall contain the following elements:

Attribute	Type	Meaning
transmitted_count	UNSIGNED32	Number of frames transmitted by this node
received_count	UNSIGNED32	Number of frames received with no errors by this node
errors_count	UNSIGNED16	Number of erroneous frames received
timeouts_count	UNSIGNED16	Number of elapsed time-outs when response expected

NOTE These counters wrap-around when the maximum value is reached, their initial value is unspecified.

4.8.4.22.3 Type_LLStatisticData

A data structure of Type_LLStatisticData shall contain the following elements:

Attribute	Type	Meaning
basic_period_count	UNSIGNED32	incremented for each Basic Period
inauguration_count	UNSIGNED16	incremented for each new inauguration
topography_count	UNSIGNED16	incremented for each new topography
transmitted_md_count	UNSIGNED32	incremented for each sent Message Data Response
received_md_count	UNSIGNED32	incremented for each received Message Data Response
line_status_a1	Type_LineStatus	See type
line_status_a2	Type_LineStatus	See type
line_status_b1	Type_LineStatus	See type
line_status_b2	Type_LineStatus	See type
line_switch_count	UNSIGNED32	incremented for each switchover to the redundant line.

NOTE These counters wrap-around when the maximum value is reached, their initial value is unspecified.

4.8.4.23 Get Inauguration Data service

Action	Return pointer to inauguration data	
Syntax	<pre> LS_T_RESULT void * * </pre>	<pre> ls_t_GetInaugData (p_inaug_data_list); </pre>
Output	p_inaug_data_list	pointer to inauguration_data of all named nodes

5 Train Network Management

This clause applies to a TCN which uses the WTB and/or the MVB.

An introduction to the Train Network Management is given in Clause A.5.

5.1 General

5.1.1 Contents of this clause

Train Network Management specifies a number of services to assist commissioning, testing, operation and maintenance of a Train Communication Network, such as:

- a) station identification and control;
- b) management of the Train Bus and Vehicle Bus link layers;
- c) distribution of routing and topography;
- d) remote reading and forcing of variables;
- e) downloading and uploading.

All services can be requested remotely by a Manager process.

These services are executed by each Station through an Agent process. This clause specifies the local managed objects and how the Agent interfaces to them.

This clause specifies the management messages exchanged between Manager and Agent for the purpose of network management.

In addition, this clause specifies how user-defined services can be interfaced to the Agent.

NOTE 1 Train Network Management does not specify application-specific messages (such as train diagnostics in accordance with UIC leaflet 557 or vehicle identification in accordance with UIC leaflet 556). However, user applications may use management services during regular operation.

NOTE 2 Train Network Management does not consider management services directly related to the application. In particular, the Virtual Device Description, which applies to the actual role of the equipment (e.g. air conditioning), is not part of this Clause.

5.1.2 Structure of this clause

This Clause is structured in the following way:

Subclause 5.1 General

Subclause 5.2 Manager, Agents and interfaces

Subclause 5.3 Managed objects

For each object, the following is specified:

- object description;
- access;
- services offered by the object.

Subclause 5.4 Services and management messages

For each service, the following is specified:

- its description;
- Call_Message and parameters;
- Reply_Message and parameters.

Subclause 5.5 Interface Procedures

For each interface, the procedures to access and control the Agent locally are specified.

5.2 Manager, Agents and interfaces

5.2.1 Manager and Agent

The Network Management services shall be provided in each Station by an Agent.

The Agent shall be identified by the Station_Id of the Station on which it resides.

The Network Management services shall be requested by a Manager.

5.2.2 Management messages protocol

For the purpose of Network Management, Manager and Agents communicate over the network by exchanging management messages, using the Messages Services of the Train Communication Network, as illustrated in Figure 211.

The Manager shall act as a Caller and the Agent as a Replier.

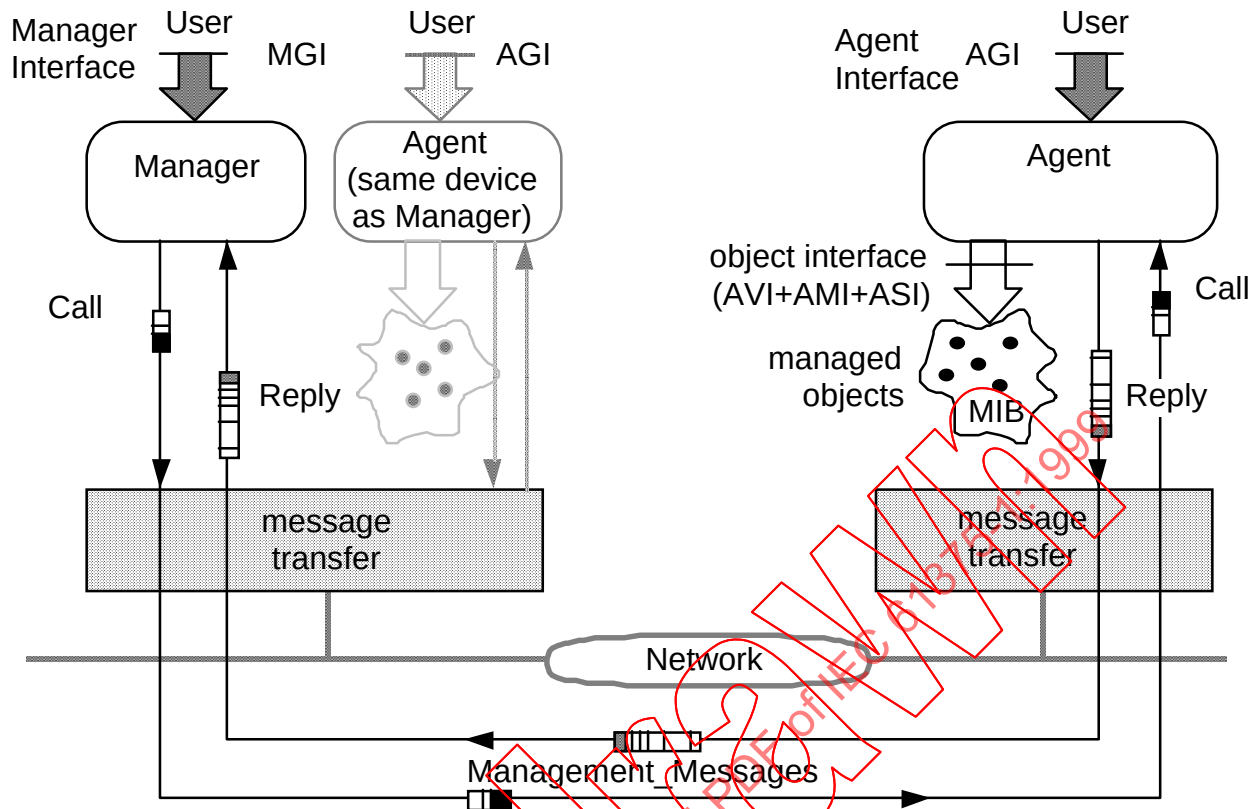


Figure 211 - Management messages

The Manager accesses a remote object in two steps.

- a) the Manager sends a management Call_Message;
- b) the Agent decodes the message, accesses the actual object and sends back a management Reply_Message, with the result of the service.

The Agent shall be accessible via both a System Address and a User Address, as Function_Id = 253.

The Manager shall be accessible via both a System Address and a User Address, as Function Id = 254.

The format of the management messages shall be as defined in the following Subclauses.

5.2.3 Interfaces

5.2.3.1 Object interface

Communication objects are related to the network communication, while non-communication objects are related to other properties of a Station.

EXAMPLES

- 1- The configuration of a Bus Administrator is a communication object.
- 2 - The memory regions or domains, the scheduler, the clock, the Station configuration and descriptors are non-communication objects.

The Agent shall access communication objects through the interfaces defined for general access in this Standard, and in particular through the:

- a) AVI (Application_Variables_Interface) for Variables;
- b) AMI (Application_Messages_Interface) for Messages;
- c) ASI (Application_Supervisory_Interface) for objects not accessible by user processes.

The ASI provides access to objects located in the different communication layers. The Agent accesses these objects through the Layer Management Interface of the layer in which they reside (Figure 212). The definition of this interface is included in the Clauses which specify the corresponding layer: Clause 2 (RTP), Clause 3 (MVB) and Clause 4 (WTB).

NOTE The entity which effectively accesses the objects in each layer is called the Layer Management Entity, or LME.

The Agent shall also be able to access non-communication objects. The interface for doing this is not specified.

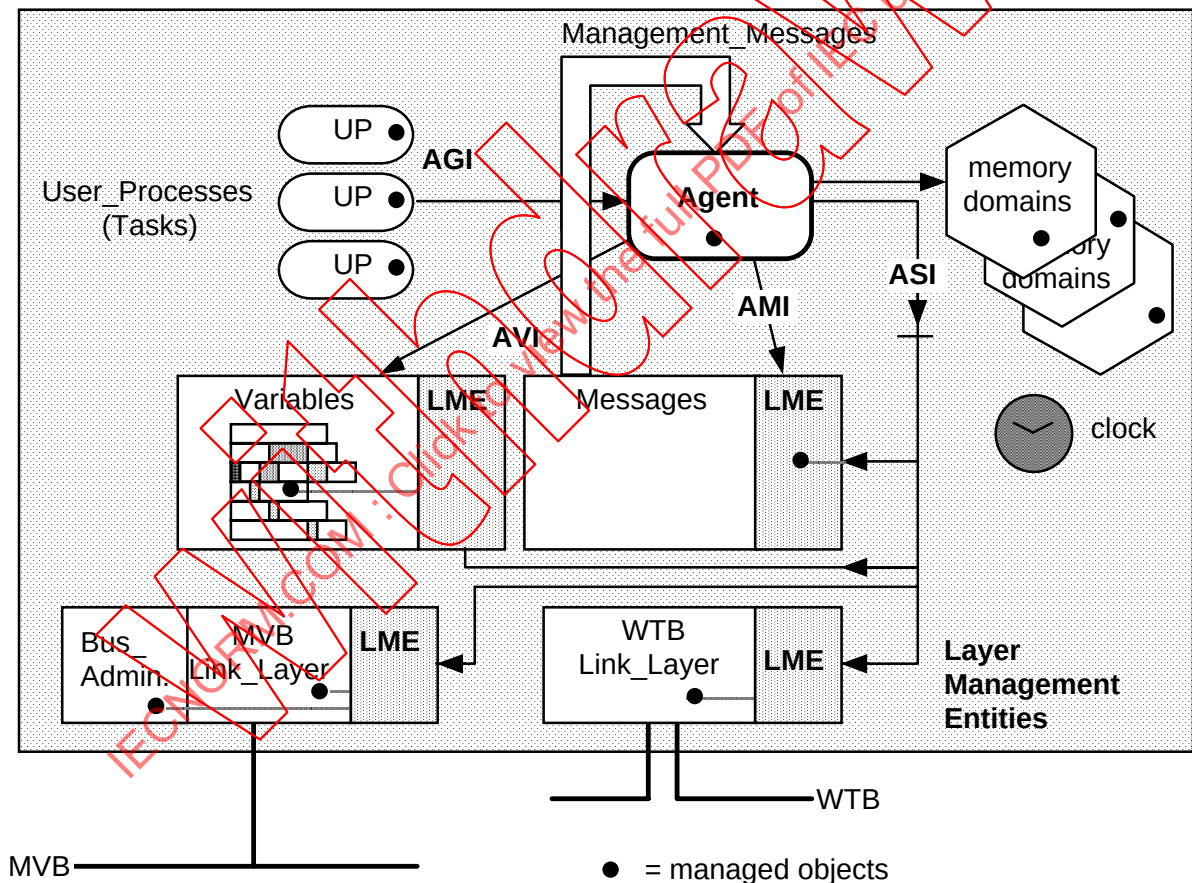


Figure 212 - Agent Interface on a (gateway) Station

5.2.3.2 Manager interface (MGI)

The Manager shall offer of a set of procedures to access remote objects, which form the Manager Interface or MGI (see Figure 211).

To each service of the Manager Interface MGI shall correspond a management Call_Message sent by the Manager to which the Agent shall reply with a management Reply_Message.

The procedures of the MGI are not individually specified, but two generic procedures are defined which provides all services.

The format of the Management Message shall also be used for the parameters of the Manager service procedure.

5.2.3.3 Agent interface (AGI)

Some managed objects need to access the Agent. This is the case for tasks which need to inquire the state of the Agent for synchronisation or for reporting status changes.

To this effect, the Agent shall provide an interface called the Agent interface (AGI), which allows local users to read and modify directly some managed objects, as specified in 5.5.2.

NOTE There are no network messages associated with the Agent interface.

5.3 Managed objects

5.3.1 Object Attributes

Every managed object shall consist of four attributes:

- a) an object identifier, a clear text string of up to 31 characters identifying the object.
- b) a status (read-only), which indicates the state of the object;
- c) a control (write-only), which acts on the object;
- d) a parameter part (read or write), which contains modifiable parts of the object, insofar as they are not already contained within the status or control.

The attributes of an object shall be handled by services defined in this Clause or by user-defined services.

Each service defined for an object shall be described by a textual information, called the Service_Descriptor. The textual information for standard services can contain a reference to this Standard. The textual information for user-defined services is not specified.

5.3.2 Station objects

5.3.2.1 Station_Status object

Each Station shall implement a Station_Status object, indicating the general characteristics of a Station and some dynamic parameters.

When a Station has only one MVB attached and no Train Bus, the Station_Status shall be a copy of the MVB Device_Status. Otherwise, the status of the different link layers shall be OR-ed to form the Station_Status.

The Station_Status shall have the following format, as illustrated by Figure 213:

```

Station_Status ::=      RECORD
{
  station_capabilities  BITSET4
  {
    sp      (0),
    ba      (1),
    gw      (2),
    md      (3)
  },
  class_specific        WORD4,
  common_flags          BITSET8
  {
    lat      (0),
    lnd      (1),
    ssd      (2),

    sdd      (3),

    scd      (4),

    frc      (5),
    snr      (6),
    ser      (7)
  }
}

```

-- basic capabilities of a station
 -- special device
 -- MVB Bus_Administrator
 -- gateway or Train Bus node
 -- messages capability
 -- always 0
 -- unused, = 0
 -- Link Disturbed
 -- Some System Disturbance fault in the controlled process (e.g. power loss)
 -- Some Disturbance of the Device:
 -- device malfunctions (e.g. error in checksum)
 -- Some Communication Disturbance:
 -- set when the sink time supervision of any port, in any Traffic_Store of this device is triggered, reset when all configured ports operate normally.
 -- forced Station: a port of this device is forced
 -- Station Not Ready:e.g. Station not initialised
 -- Station reserved by a Manager.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sp	ba	gw	md					lat	lnd	ssd	Sdd	scd	frc	snr	ser
station_capabilities				class_specific				Common_flags							

Figure 213 - Station_Status

5.3.2.2 Station_Control object

Every Station shall implement a Station_Control object with two associated services:

- start a Station and assign configuration parameters, such as the Station's name;
- restart a Station, i.e. stop all tasks, clear all tables, close all conversations and restart only the messenger and the Agent.

If only application tasks are to be started or stopped, the task object shall permit a soft restart.

5.3.2.3 Station Inventory object

Every Station shall implement a read-only inventory object describing the static characteristics of a Station, consisting of:

a) its identification

- manufacturer,
- serial number,
- Station_Identifier and name;

b) its capabilities

- software version,
- supported services,
- list of link layers.

The list of link layers is represented by a Link_Set, a 16-bit bitset having one bit for each of the link layers. To each link layer, a Traffic_Store is associated. A Link_Set is defined as:

```
Link_Set ::= BITSET16
{
  link_layer0 (0), -- first transmitted bit
  link_layer1 (1),
  link_layer2 (2),
  link_layer3 (3),
  link_layer4 (4),
  link_layer5 (5),
  link_layer6 (6),
  link_layer7 (7),
  link_layer8 (8),
  link_layer9 (9),
  link_layer10 (10),
  link_layer11 (11),
  link_layer12 (12),
  link_layer13 (13),
  link_layer14 (14),
  link_layer15 (15)
}
```

5.3.2.4 Station Reservation object

Every Station shall implement a reservation object allowing a Manager to reserve this Station for its exclusive use.

The reservation object shall be a semaphore with time-out which ensures exclusive access to modifiable objects.

The state of that semaphore shall be reflected in the 'ser' bit of the Station_Status ("1" = reserved, "0" = not reserved).

The services on the reservation object are "reserve" and "release".

A Manager shall reserve a Station before it is allowed to modify its managed objects.

A Manager which intends to reserve a Station shall send a Reservation Request to the Agent.

An Agent which accepts a reservation shall set the reservation object and register the identity of its Manager.

The owner of the reservation object shall be identified by the 24-bit caller address received by the Agent.

In addition, the reservation call provides a 32-bit word to identify more closely the Manager (or the person which manages the network) for specific applications.

If a Manager reserves several stations, it shall reserve stations in order of increasing Application_Address to prevent deadlocks with another Manager.

An Agent shall reject any modifying service calls, if it is not reserved ('ser' bit cleared) or if the request is issued by another non-registered Manager.

The time-out ensures that a Station does not remain stuck if the Manager retires without releasing the Station.

The semaphore's time-out shall be reset each time the Agent receives a management Call_Message from its registered Manager.

The reservation semaphore shall be cleared:

- a) by an inauguration of the Train Bus;
- b) if the station is reset;
- c) by an "override" reservation message.

NOTE 1 Non-modifying services (read) are not subject to reservation. They may be called by any Manager or by any other function. Only a registered Manager may issue modifying requests (writes).

NOTE 2 The identity of both the Manager and the Agent may be affected by an inauguration.

NOTE 3 The "override" message is a last-resort method to reserve a Station which for some reason remains stuck in a reserved state. It should be used with caution.

NOTE 4 The reservation provides no access security. It can only protect against certain mishaps. If secure access is needed, it is recommended to implement passwords at the workstation level.

5.3.3 MVB link objects

5.3.3.1 MVB_Status object

Each Station attached to an MVB shall implement a read-only MVB_Status object.

The MVB_Status object shall identify the hardware and software version, define dynamic and static parameters, disclose errors and statistics about the MVB Link_Layer and contain the Device_Status specified in Clause 3 ("Multifunction Vehicle Bus").

NOTE The Agent accesses this object through the MVB link layer supervisory interface, Is_v_xxx.

5.3.3.2 MVB_Control object

Each Station with an MVB attachment supporting the Bus_Administrator capability shall implement a write-only MVB_Control object.

Over this object, a Manager can command a Bus_Administrator:

- a) to release control so that another Bus_Administrator can take over; or
- b) to use a polling list which has been loaded.

5.3.3.3 MVB_Configuration object

Each Station with an MVB attachment supporting the Bus_Administrator capability shall implement the write-only Bus_Configuration object, containing all parameters required for the Bus_Administrator to operate.

The Bus_Configuration shall comprise in particular the following sub-objects, which are treated as a whole and are covered by a common version control:

- a) The set-up parameters of the Bus_Administrator;
- b) The Known_Devices_List, which indicates which devices should normally be present;
- c) The Periodic_List, which defines the polling strategy for Process Data;
- d) The Bus_Administrators_List, which defines the mastership transfer algorithm;
- e) The Devices_List, which defines how known and unknown devices are scanned.

NOTE 1 Modifying the Periodic_List on-line can lead to a non-deterministic operation or even cause a breakdown in traffic if configuration fails (e.g. too many variables included).

NOTE 2 The mastership transfer protocol ensures that once one Bus_Administrator starts using the new configuration list, all others will do likewise.

5.3.3.4 MVB_Devices object

Each Station with an MVB attachment supporting the Bus_Administrator capability shall implement a read-only MVB_Devices object, containing the list of the present devices together with their Device_Status.

NOTE 1 The Bus_Administrator builds this list by scanning the devices periodically.

NOTE 2 This object may also be implemented by a Station which is not a Bus_Administrator, by observing the bus traffic.

5.3.4 WTB link objects

5.3.4.1 WTB_Status object

Each node attached to the Train Bus shall implement a read-only WTB_Status object, which is specified in Clause 4 of this standard ("Wire Train Bus").

The WTB_Status identifies the hardware and software version, defines dynamic and static parameters, discloses errors and statistics about the WTB Link_Layer and contains the Node Status Word corresponding to this WTB connection

The Agent accesses this object through the WTB link layer supervisory interface, ls_t_xxx.

The WTB_Status object includes the following data structures, supplied by the local application and defined in Clause 4:

- a) Node_Descriptor, a structure defining the Process_Data frame, (see 4.7.2.1)
- b) Node_Report, a BITSET8 which signals disturbances, (see 4.7.2.2)
- c) User_Report, a WORD8 supplied by the application (see 4.7.2.3)
- d) Node_Key, a RECORD containing the Node_Type and Node_Version (see 4.7.2.4))
- e) Ls_T_State, an ENUM8 which indicates in which states a node is (see 4.7.2.5)
- f) Node_Strength, an ENUM8 expressing the strength of the node (see 4.8.4.16.3)

5.3.4.2 WTB Topography object

Each node attached to the Train Bus shall implement a read-only WTB_Topography object, which is specified in Clause 4 ("Wire Train Bus").

The Agent accesses this object through the link management interface, Is t xxx.

NOTE 1 The WTB Topography describes the current Train Bus configuration, lists the existing nodes and their descriptors, and reports the master status.

NOTE 2 The Topography object should be accessed by all applications which send messages over the Train Bus, since the topography ensures consistency of addressing when the composition changes, as described in 2.3.

NOTE 3 The topography can be obtained from any Station. However, only a Station with a Train Bus connection can give the current version of the topography.

NOTE 4 The topography object consists of a mandatory part and of the user-supplied Inauguration_Data.

5.3.5 ~~Variable objects~~

5.3.5.1 Port Configuration object

~~Each Station with Process_Data capability shall implement the read-only Ports_Configuration object, which indicates the number of ports and their addresses.~~

A port may be of fixed length (MVB) or of variable length (WTB).

The length of a port is defined by its F code:

```
F_Code ::=          ENUM4
{
  PD16      (0),      -- 16-bit port
  PD32      (1),      -- 32-bit port
  PD64      (2),      -- 64-bit port
  PD128     (3),      -- 128-bit port
  PD256     (4),      -- 256-bit port
  PD512     (5),      -- 512-bit port
  PD1024    (6),      -- 1024-bit port
  PDW       (7)       -- variable size port (WTB)
}
```

A port is defined on a bus by its 12-bit port address. On the WTB, only the lower 6 bits are used. The F_code and address of a port are combined in a 16-bit structure called FcodeAdr:

```
FcodeAdr ::=
{
  f_code          F_Code,          -- F_code giving the port size
  address         UNSIGNED12       -- logical address
}
```

Each port is characterised by the following attributes:

```
Port_Object ::= RECORD
{
  f_code          F_Code          -- F_code of the port
  port_address    UNSIGNED12      -- address of the port
  port_config     BITSET4
  {
    src           [0]             -- port enabled as publisher
                                   (source)
    snk           [1]             -- port enabled as subscriber
                                   (sink)
    twc           [2]             -- port transfer frames with
                                   checksum
    frc           [3]             -- port forced to force value
  }
  port_size       UNSIGNED8       -- maximum port size in octets,
                                   only for PDW (WTB port)
}
```

5.3.5.2 Variables object

5.3.5.2.1 Object

Each Station with Process_Data capability shall implement the Variables object, indicating the value, check variable and freshness of a Variable.

NOTE The individual period at which a port is refreshed is controlled by the Bus_Configuration object on the Vehicle Bus or by the medium access control of the node on the Train Bus. The association between ports, variables and bus addresses is an application issue.

5.3.5.2.2 Access

As any other application process, the Agent accesses Variables through the Application Layer Interface for Variables (AVI). This interface allows the reading and forcing of variables in the Traffic_Store, either individually or by Clusters.

5.3.5.2.3 Services

All remote services on variables are performed on Clusters (multiple variables), the individual variable access being a special case.

The following services access the variables remotely:

- a) read: retrieve the value of a variable, its check variable and freshness.
- b) force: force a variable to a specified value. This value cannot be overwritten by an application (source) or by the bus (sink) until the variable is unforced. Other variables of the same Dataset can be forced or unforced.

- Since a forced variable cannot be overwritten by its producer, the Agent shall request over the AGI permission to modify it.
 - If the variable is located in a source port, the bus will broadcast its value to all sinks starting with the next period.
 - If the variable is located in a sink port, the forced value will be only visible to its own Station. Bus writing to this variable is inhibited while the variable is forced, but local freshness is still controlled by the bus.
 - If any variable in a Station has been forced, the Agent sets the "frc" bit in its Station_Status and in the Device_Status of the MVB connected to the forced Traffic_Store.
- c) **unforce**: releases a forced variable so that its application can modify its value again.
- d) **unforce_all**: releases all variables in a Traffic_Store.
- e) **read_bindings**: indicates to which port a local variable is bound.
- f) **write_bindings**: binds a local variable to a port.

NOTE 1 It is possible for a user process to access a variable by requesting its local Agent to carry out a read_variables or a force_variables on one of its own Traffic_Stores. This, however, takes much more time than the direct access.

NOTE 2 A consumer application which needs to distinguish a forced from a non-forced variable may set the check variable of a forced variable to be equal to '10'B.

NOTE 3 Unforce_all should be called once for each Traffic_Store until the FRC-bit in the Station_Status is reset.

5.3.5.3 Port_Attach

Each MVB Class 2 device shall implement the Port_Attach object, which defines the marshalling between the ports and the input/output points. This object may only be remotely written.

5.3.6 Messenger objects

5.3.6.1 Messenger status

Each Station providing the Messages service shall implement the read-only Messenger status object, which reports the version of the software, configuration and usage statistics of the two protocols:

- a) the single-cast protocol; and
- b) the (optional) multicast protocol.

The Agent accesses this object through the ASI interface of the Messenger

5.3.6.2 Messenger control

Each Station providing the Messages service shall implement the Messenger control object, which allows the remote configuration of Message communication parameters.

5.3.6.3 Directory objects

Each Station providing the Messages service shall implement the Function_Directory, which indicates for each Function which Station executes it.

Each Station with Messages capability, but not using simple routing, shall implement the Station_Directory, which maps the Station_Identifier to Next_Station and its physical address (Bus_Id and Device_Address).

Each Terminal Station supporting the multicast protocol and each Router Station shall implement the Group_Directory object, which indicates to a Station which groups it belongs to.

Each Node of a fixed configuration Train Bus with Messages capability, but not using simple routing, shall implement the Node_Directory, which indicates which device address corresponds to a given Node address.

There shall exist two services for each directory:

- a) read_xxx_directory, which reads the whole directory;
- b) write_xxx_directory, which writes the whole directory (with or without clearing it beforehand).

NOTE 1 The Station_Directory can be replaced by simple routing in terminal stations, in which case no Station_Directory object is provided.

NOTE 2 The Agent accesses directories through the AMI interface, as in any other application.

5.3.7 Domain objects

5.3.7.1 Objects

Each Station with the ability of uploading or downloading memory regions shall implement the Domains object.

The number of domains per station is not specified.

For technological reasons, the domains should be wholly contained within the same memory type.

As a special case, the Manager may access physical memory regions of a domain.

NOTE 1 Domains are memory regions containing code or data. Domains may be located indifferently in memories of different technology, RAM, EEPROM or Flash-EPROM.

NOTE 2 The Agent accesses domains through implementation-specific procedures, which provide reading/writing of memory locations of different technology, possibly protected by access rights or memory management.

NOTE 3 Access to domains requires a precise knowledge of the configuration of a Station.

5.3.7.2 Services

The services on Domain objects shall comprise:

- a) set-up download (prepare download, verify, boot);
- b) download segment;
- c) read memory;
- d) write memory.

NOTE 1 Domain loading may overwrite existing regions or even the communication software and the Agent itself.

NOTE 2 Before downloading a domain, it is safer for the Manager to issue a call "reset_station" first. In this case, the Manager should reserve the Station once more to download it.

5.3.8 Task objects

5.3.8.1 Objects

Each Station capable of managing user tasks shall implement the write-only Task object.

The number of tasks is not specified.

The tasks are treated as a whole for the purpose of management.

The task control object controls the execution of all tasks.

5.3.8.2 Access

The Agent is assumed to access the scheduler to start and stop tasks.

The access to tasks is implementation-dependent.

5.3.8.3 Services

Three task services are specified:

- a) start, which starts all tasks;
- b) stop, which stops all tasks.
- c) tasks_list, which reads the list of tasks installed in the station and their status.

Before stopping a task, the Agent shall get the task's permission through the AGI (see 5.2.3.3) except when "override" access is specified.

5.3.9 Clock object

5.3.9.1 Objects

Each Station with a remotely accessible clock shall implement the Clock object.

The clock accuracy is not prescribed.

NOTE 1 Setting or reading the clock through a Management_Message is subject to unpredictable delays. Especially, an upper limit for the delivery of a message cannot be given since it depends on the presence of other messages in the queue of this and of the other stations.

NOTE 2 A more precise clock setting can be obtained by letting the Bus_Administrator send a synchronisation variable at a determined time. This is an implementation issue.

5.3.9.2 Services

Two clock services are specified:

- a) read_clock, which reads the current time value;
- b) set_clock, which sets the clock value.

5.3.10 Journal object

5.3.10.1 Object

A Station may implement for debugging purposes a journal object

The journal consists of a list of user-defined entries with a serial number and a time-stamp. The journal is expected to be implemented as a circular buffer in which the last “j” entries are registered, older ones being discarded.

The journal may be read by several managers, reading is not a consuming action.

The journal may be written in parallel at high speed by applications on the node. To avoid buffering, an index scheme allows to select the valid portion of the journal.

Each task may enter events into the journal through the AGI interface.

5.3.10.2 Services

One journal service is specified:

- read_journal, which reads the last entries of the journal.

5.3.11 Equipment object

5.3.11.1 Object

A Station may implement an Equipment Descriptor which describes the supported equipment.

5.3.11.2 Services

One service is specified:

- read_equipment_descriptor, which retrieves a pointer to the memory location where the equipment descriptor is located

5.4 Services and management messages

5.4.1 Notation for all management messages

5.4.1.1 Message structure

Each service shall be invoked by a Call/Reply Management message exchange with the following format:

```

Management_Message ::= RECORD
{
    tnm_key          ENUM8          -- first octet
    {
        CALL      ('02'H),          -- Call (request)
        REPLY     ('82'H)          -- Reply (response)
    },
    message          ONE_OF [tnm_key] -- selects call or reply
    {
        [CALL]      Call_Mgt_Message, -- described below
        [REPLY]     Reply_Mgt_Message -- described below
    }
}
    
```

The most significant bit of 'tnm_key' shall indicate if this is a Call or a Reply message.

For private Network Management, values of 'tnm_key' shall be in the range '40'H -'7F'H (Call), or 'C0'H - 'FF'H (Reply). Other values are reserved for future use.

For the default allocation of `tnm_keys`, the UIC company code may be added to '40'H to form the `tnm_key`.

NOTE Fields in management messages have been aligned on a 32-bit boundary to speed up operation with 32-bit processors. This alignment assumes that the first field, 'tnm_key', is located at an address divisible by four. The message service should respect this convention for sending as well as for receiving.

5.4.1.2 Notation for the SIF_code

The `SIF_code` indicates the requested service. The least significant bit indicates if this is a read or a write (modifying) service. Read services may be used without previous reservation. The following `SIF_codes` are defined for the default `tnm_key` pair ('02'H / '82'H).

```
Sif_Code ::= ENUM8      -- choice of the service
{
  READ_STATION_STATUS      (00),
  WRITE_STATION_CONTROL    (01),
  READ_STATION_INVENTORY   (02),
  WRITE_RESERVATION        (03),
  READ_SERVICE_DESCRIPTOR  (04),
  READ_LINKS_DESCRIPTOR    (06),
  WRITE_LINKS_DESCRIPTOR   (07),
  READ_MVB_STATUS          (10),
  WRITE_MVB_CONTROL        (11),
  READ_MVB_DEVICES         (12),
  WRITE_MVB_ADMINISTRATOR  (13),
  READ_WTB_STATUS          (20),
  WRITE_WTB_CONTROL        (21),
  READ_WTB_NODES           (22),
  READ_WTB_TOPOGRAPHY      (24),
  WRITE_ATTACH_PORT        (29),
  READ_PORTS_CONFIGURATION (30),
  WRITE_PORTS_CONFIGURATION (31),
  READ_VARIABLES           (32),
  WRITE_FORCE_VARIABLES    (33),
  WRITE_UNFORCE_VARIABLES  (35),
  WRITE_UNFORCE_ALL        (37),
  READ_VARIABLE_BINDINGS   (38),
  WRITE_VARIABLE_BINDINGS  (39),
  READ_MESSENGER_STATUS    (40),
  WRITE_MESSENGER_CONTROL  (41),
  READ_FUNCTION_DIRECTORY  (42),
  WRITE_FUNCTION_DIRECTORY (43),
  READ_STATION_DIRECTORY   (44),
  WRITE_STATION_DIRECTORY  (45),
  READ_GROUP_DIRECTORY     (46),
  WRITE_GROUP_DIRECTORY    (47),
  READ_NODE_DIRECTORY      (48),
  WRITE_NODE_DIRECTORY     (49),
  READ_MEMORY              (50),
  WRITE_MEMORY             (51),
  WRITE_DOWNLOAD_SETUP     (53),
  WRITE_DOWNLOAD_SEGMENT   (55),
  READ_TASKS_STATUS        (60),
  WRITE_TASKS_CONTROL      (61),
  READ_CLOCK               (70),
  WRITE_CLOCK              (71),
  READ_JOURNAL             (80),
  READ_EQUIPMENT           (82),
  USER_SERVICE_0          (128),
  USER_SERVICE_127        (255)
}

-- (128..255) reserved for user services.
```

5.4.1.3 Notation for a call management message

```

Call_Mgt_Message ::= RECORD
{
  sif_code          Sif_Code,          -- the second octet is the
                                         SIF_code
  message_body      ONE_OF [sif_code]
  {
    [READ_STATION_STATUS]      Call_Read_Station_Status,
    [WRITE_STATION_CONTROL]    Call_Write_Station_Control,
    [READ_STATION_INVENTORY]   Call_Read_Station_Inventory,
    [WRITE_RESERVATION]        Call_Write_Reservation,
    [READ_SERVICE_DESCRIPTOR]   Call_Read_Service_Descriptor,
    [READ_LINKS_DESCRIPTOR]     Call_Read_Links_Descriptor,
    [WRITE_LINKS_DESCRIPTOR]    Call_Write_Links_Descriptor,
    [READ_MVB_STATUS]          Call_Read_Mvb_Status,
    [WRITE_MVB_CONTROL]        Call_Write_Mvb_Control,
    [READ_MVB_DEVICES]         Call_Read_Mvb_Devices,
    [WRITE_MVB_ADMINISTRATOR]   Call_Write_Mvb_Administrator,
    [READ_WTB_STATUS]          Call_Read_Wtb_Status,
    [WRITE_WTB_CONTROL]        Call_Write_Wtb_Control,
    [READ_WTB_NODES]           Call_Read_Wtb_Nodes,
    [READ_WTB_TOPOGRAPHY]      Call_Read_Wtb_Topography,
    [WRITE_ATTACH_PORT]        Call_Write_Attach_Port,
    [READ_PORTS_CONFIGURATION]  Call_Read_Ports_Configuration,
    [WRITE_PORTS_CONFIGURATION] Call_Write_Ports_Configuration,
    [READ_VARIABLES]           Call_Read_Variables,
    [WRITE_FORCE_VARIABLES]     Call_Write_Force_Variables,
    [WRITE_UNFORCE_VARIABLES]   Call_Write_Unforce_Variables,
    [WRITE_UNFORCE_ALL]         Call_Write_Unforce_All,
    [READ_VARIABLE_BINDINGS]    Call_Read_Variable_Bindings,
    [WRITE_VARIABLE_BINDINGS]   Call_Write_Variable_Bindings,
    [READ_MESSENGER_STATUS]     Call_Read_Messenger_Status,
    [WRITE_MESSENGER_CONTROL]   Call_Write_Messenger_Control,
    [READ_FUNCTION_DIRECTORY]   Call_Read_Function_Directory,
    [WRITE_FUNCTION_DIRECTORY]  Call_Write_Function_Directory,
    [READ_STATION_DIRECTORY]    Call_Read_Station_Directory,
    [WRITE_STATION_DIRECTORY]   Call_Write_Station_Directory,
    [READ_GROUP_DIRECTORY]      Call_Read_Group_Directory,
    [WRITE_GROUP_DIRECTORY]     Call_Write_Group_Directory,
    [READ_NODE_DIRECTORY]       Call_Read_Node_Directory,
    [WRITE_NODE_DIRECTORY]      Call_Write_Node_Directory,
    [READ_MEMORY]               Call_Read_Memory,
    [WRITE_MEMORY]              Call_Write_Memory,
    [WRITE_DOWNLOAD_SETUP]      Call_Write_Download_Setup,
    [WRITE_DOWNLOAD_SEGMENT]    Call_Write_Download_Segment,
    [READ_TASKS_STATUS]         Call_Read_Tasks_Status,
    [WRITE_TASKS_CONTROL]       Call_Write_Tasks_Control,
    [READ_CLOCK]                Call_Read_Clock,
    [WRITE_CLOCK]               Call_Write_Clock,
    [READ_JOURNAL]              Call_Read_Journal,
    [READ_EQUIPMENT]            Call_Read_Equipment,
  }
}

```

5.4.1.4 Notation for a reply management message

```

Reply_Mgt_Message ::= RECORD
{
  sif_code          Sif_Code,          -- the second octet is the
                                         SIF_code
  message_body      ONE_OF [sif_code]
  {
    [READ_STATION_STATUS]          Reply_Read_Station_Status,
    [WRITE_STATION_CONTROL]        Reply_Write_Station_Control,
    [READ_STATION_INVENTORY]       Reply_Read_Station_Inventory,
    [WRITE_RESERVATION]            Reply_Write_Reservation,
    [READ_SERVICE_DESCRIPTOR]      Reply_Read_Service_Descriptor,
    [READ_LINKS_DESCRIPTOR]        Reply_Read_Links_Descriptor,
    [WRITE_LINKS_DESCRIPTOR]       Reply_Write_Links_Descriptor,
    [READ_MVB_STATUS]              Reply_Read_Mvb_Status,
    [WRITE_MVB_CONTROL]            Reply_Write_Mvb_Control,
    [READ_MVB_DEVICES]             Reply_Read_Mvb_Devices,
    [WRITE_MVB_ADMINISTRATOR]      Reply_Write_Mvb_Administrator,
    [READ_WTB_STATUS]              Reply_Read_Wtb_Status,
    [WRITE_WTB_CONTROL]            Reply_Write_Wtb_Control,
    [READ_WTB_NODES]               Reply_Read_Wtb_Nodes,
    [READ_WTB_TOPOGRAPHY]          Reply_Read_Wtb_Topography,
    [WRITE_ATTACH_PORT]            Reply_Write_Attach_Port,
    [READ_PORTS_CONFIGURATION]     Reply_Read_Ports_Configuration,
    [WRITE_PORTS_CONFIGURATION]    Reply_Write_Ports_Configuration,
    [READ_VARIABLES]               Reply_Read_Variables,
    [WRITE_FORCE_VARIABLES]        Reply_Write_Force_Variables,
    [WRITE_UNFORCE_VARIABLES]      Reply_Write_Unforce_Variables,
    [WRITE_UNFORCE_ALL]            Reply_Write_Unforce_All,
    [READ_VARIABLE_BINDINGS]       Reply_Read_Variable_Bindings,
    [WRITE_VARIABLE_BINDINGS]      Reply_Write_Variable_Bindings,
    [READ_MESSENGER_STATUS]        Reply_Read_Messenger_Status,
    [WRITE_MESSENGER_CONTROL]      Reply_Write_Messenger_Control,
    [READ_FUNCTION_DIRECTORY]      Reply_Read_Function_Directory,
    [WRITE_FUNCTION_DIRECTORY]     Reply_Write_Function_Directory,
    [READ_STATION_DIRECTORY]       Reply_Read_Station_Directory,
    [WRITE_STATION_DIRECTORY]      Reply_Write_Station_Directory,
    [READ_GROUP_DIRECTORY]         Reply_Read_Group_Directory,
    [WRITE_GROUP_DIRECTORY]        Reply_Write_Group_Directory,
    [READ_NODE_DIRECTORY]          Reply_Read_Node_Directory,
    [WRITE_NODE_DIRECTORY]         Reply_Write_Node_Directory,
    [READ_MEMORY]                  Reply_Read_Memory
    [WRITE_MEMORY]                 Reply_Write_Memory
    [WRITE_DOWNLOAD_SETUP]         Reply_Write_Download_Setup,
    [WRITE_DOWNLOAD_SEGMENT]       Reply_Write_Download_Segment,
    [READ_TASKS_STATUS]            Reply_Read_Tasks_Status,
    [WRITE_TASKS_CONTROL]          Reply_Write_Tasks_Control,
    [READ_CLOCK]                   Reply_Read_Clock,
    [WRITE_CLOCK]                  Reply_Write_Clock,
    [READ_JOURNAL]                 Reply_Read_Journal,
    [READ_EQUIPMENT]               Reply_Read_Equipment
  }
}

```

5.4.1.5 Status and Error report

The Agent shall report success or failure through its replier status, which is not transmitted within the reply message body, but passed as a separate parameter in the Session_Header.

The result set by the Agent may be modified by the network if a communication error occurs. In this case, the network returns a communication error, by replacing the Agent code by its own Am_Result.

The replier status shall be returned as a parameter of "mm_service_conf" to the Manager.

The status and error report shall combine the results of the agent and of the network, as specified by the Mm_Result type:

Mm_Result ::=	ENUM8	
{		
MM_OK	(0)	-- successful termination
AM_FAILURE	(1)	-- unspecified failure
AM_BUS_ERR	(2)	-- no bus transmission possible
AM_REM_CONN_OVF	(3)	-- too many incoming connections
AM_CONN_TMO_ERR	(4)	-- Connect Request not answered
AM_SEND_TMO_ERR	(5)	-- time-out SEND_TMO elapsed
AM_REPLY_TMO_ERR	(6)	-- no reply received
AM_ALIVE_TMO_ERR	(7)	-- time-out ALIVE_TMO elapsed
AM_NO_LOC_MEM_ERR	(8)	-- not enough memory or timers
AM_NO_REM_MEM_ERR	(9)	-- no more memory or timers at partner
AM_REM_CANC_ERR	(10)	-- cancelled by partner
AM_ALREADY_USED	(11)	-- same operation already done
AM_ADDR_FMT_ERR	(12)	-- address format error
AM_NO_REPLY_EXP_ERR	(13)	-- no such reply expected
AM_NR_OF_CALLS_OVF	(14)	-- too many calls requested
AM_REPLY_LEN_OVF	(15)	-- Reply_Message too long
AM_DUPL_LINK_ERR	(16)	-- duplicated conversation error
AM_MY_DEV_UNKNOWN_ERR	(17)	-- my device unknown
AM_NO_READY_INST_ERR	(18)	-- no ready Replier instance
AM_NR_OF_INST_OVF	(19)	-- too many Replier instances
AM_CALL_LEN_OVF	(20)	-- Call_Message too long
AM_UNKNOWN_DEST_ERR	(21)	-- partner device unknown
AM_INAUG_ERR	(22)	-- train inauguration occurred
AM_TRY_LATER_ERR	(23)	-- (internally used only)
AM_FIN_NOT_REG_ERR	(24)	-- final address not registered
AM_GW_FIN_NOT_REG_ERR	(25)	-- final address not registered in router
AM_GW_ORI_REG_ERR	(26)	-- origin address not registered in router
MM_SIF_NOT_SUPPORTED	(33)	-- SIF_code not supported
MM_RDONLY_ACCESS	(34)	-- read only access permitted
MM_CMD_NOT_EXECUTED	(35)	-- service failed
MM_DNLD_NO_FLASH	(36)	-- no non-volatile storage at that location
MM_DNLD_FLASH_HW_ERR	(37)	-- hardware error during download
MM_BAD_CHECKSUM	(38)	-- domain has an incorrect checksum
MM_INT_ERROR	(39)	-- internal error
MM_ER_VERS	(40)	-- erroneous version
MM_BUS_HW_BAD	(41)	-- link damaged
MM_BUS_HW_NO_CONFIG	(42)	-- unconfigured hardware
MM_LP_ERROR	(43)	-- failed access to Traffic_Store
MM_VERSION_CONFLICT	(44)	-- version conflict
}		

5.4.2 Station services

5.4.2.1 Read_Station_Status

5.4.2.1.1 Description

This service reads the Station_Status object remotely.

5.4.2.1.2 Call_Read_Station_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 0							

```
Call_Read_Station_Status ::= RECORD
{
    -- no parameters
}
```

5.4.2.1.3 Reply_Read_Station_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 0							
bus_id								reserved1							
								device_address							
								station_status							

```
Reply_Read_Station_Status ::= RECORD
{
    bus_id          UNSIGNED8 (0..15), -- identifier of the link (e.g.
                                         MVB1, WTB) over which the
                                         agent received the call.
                                         This link may not change for
                                         the whole management
                                         session.

    reserved1       WORD8 (≠0),        -- reserved
    device_address  WORD16,             -- device_address of the bus
                                         over which the agent
                                         received the call
    station_status  Station_Status     -- see definition
}
```

NOTE A Manager may gain access to an unknown Station by accessing it over its Device_Address.

5.4.2.2 Write_Station_Control

5.4.2.2.1 Description

This service resets a Station and assigns its Station name and Station_Identifier.

5.4.2.2.2 Call_Write_Station_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 1							
command							RST	station_id							
station_name: STRING32															
(CHARACTER8)								CHARACTER8 or '00'H							

Call_Write_Station_Control ::= RECORD

```

{
  command          BITSET8
  {
    rst            (7)
    -- if 1: reset the Station,
    -- clear all tables and the
    -- reservation semaphore and
    -- start only messenger and
    -- Agent.
    -- if 0: assign only the new
    -- name and Station_Identifier.

  },
  station_id       UNSIGNED8,
  -- 8-bit Station_Identifier
  -- assigned to this Station.
  -- If station_id = 0, the
  -- Station_Identifier is not
  -- changed.

  station_name     STRING32
  -- name assigned to this
  -- Station. If the size of
  -- station_name is zero, the
  -- name of the Station is not
  -- changed.
}

```

5.4.2.2.3 Reply_Write_Station_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 1							
bus_id								reserved1							
device_address															

```

Reply_Write_Station_Control ::= RECORD
{
  bus_id          UNSIGNED8 (0..15),
  -- link over which the Agent
  -- received the Call_Message

  reserved1       WORD8 (=0),
  -- reserved

  device_address  WORD16
  -- address over which the Agent
  -- received the Call_Message
}

```


link_set	Link_Set	-- one bit for each supported traffic stores or link layers, each link layer corresponding to a Traffic_Store, the first bit corresponds to Traffic_Store 0.
reserved2	WORD8 (=0),	-- reserved
station_id	UNSIGNED	-- identifier of this Station (0 or 'FF'H if undefined)
station_name	STRING32	-- name given to this Station (initially void)
station_status	Station_Status	-- Station Status Word
}		

NOTE A Manager may gain access to an unknown Station by accessing it over its Device_Address.

5.4.2.4 Write_Station_Reservation

5.4.2.4.1 Description

This service accesses the reservation object, reserves or releases the Station.

The reservation call specifies the Manager (through its Application_Address), the reservation time-out and the access rights. It includes a 32-bit Manager identifier, the use of which is application-dependent.

A Station shall reject the reservation request if it is already reserved and the Manager is different from the current one.

A Station which is reserved shall set the 'ser' bit in its Station_Status and in the Device_Status of every MVB connected to it.

The reservation time-out shall be restarted by the reception of any service call coming from the Manager in charge.

The Station shall release the reservation and clear the 'ser' bits:

- a) when a "release" call is received, without the option of restarting (normal case);
- b) when the reservation time-out expires;
- c) when a train inauguration takes place and the Manager is located on another node;
- d) when a "reset" is received;
- e) when an "override" is received.

In the last two cases, the Station shall execute the procedure subscribed by the application as "station_restart" (see 5.5.2.4).

5.4.2.5 Read_Service_Descriptor

5.4.2.5.1 Description

This service reads the Service_Descriptor, the description text defining a service or the object accessed by this service. It is normally used only for user-defined services.

5.4.2.5.2 Call_Read_Service_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 4							
reserved1								get_sif_code							

```

Call_Read_Service_Descriptor ::= RECORD
{
  reserved1          WORD8 (=0),          -- reserved
  get_sif_code       Sif_Code             -- service to be described
}
  
```

5.4.2.5.3 Reply_Read_Service_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 4							
reserved1								get_sif_code							
								reserved2							
								string_size							
service_description: ARRAY ALIGN16 [string_size] OF															
(CHARACTER8)								CHARACTER8 or '00'H							

```

Reply_Read_Service_Descriptor ::= RECORD
{
  reserved1          WORD8 (=0),          -- reserved
  get_sif_code       Sif_Code,            -- service to be described
  reserved2          WORD16 (=0),         -- reserved
  string_size        UNSIGNED16,         -- up to 65535 characters
  service_description ARRAY ALIGN16 [string_size] OF
    CHARACTER8      -- user-defined string
}
  
```

5.4.2.6 Read_Links_Descriptor

5.4.2.6.1 Description

This service reads the Links_Descriptor, the description text which shall describe each link layer present in the Station_Inventory.

5.4.2.6.2 Call_Read_Links_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 6							

```

Call_Read_Links_Descriptor ::= RECORD
{
  -- no parameters
}
  
```



```

Call_Write_Links_Descriptor ::= RECORD
{
  nr_links          UNSIGNED16 (0..15), -- number of supported links
  links_descriptor  ARRAY [nr_links] OF
  {
    bus_id          UNSIGNED8 (0..15), -- identifier of the link
    link_type       ENUM8
    {
      LINK_UNKNOWN  (0), -- unknown
      LINK_MVB      (1), -- MVB bus
      LINK_WTB      (2), -- WTB bus
      LINK_MBX      (3), -- memory mailbox
      LINK_SER      (4), -- serial link
                        -- other reserved
    },
    link_name       STRING32 -- name of the link
  }
}

```

5.4.2.7.3 Reply_Write_Links_Descriptor

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 7							

```

Reply_Write_Links_Descriptor ::= RECORD
{
} -- no parameters

```

5.4.3 MVB link services

5.4.3.1 Read_MVB_Status

5.4.3.1.1 Description

The service reads the MVB_Status, which is the status of an MVB link layer to which a Station is connected.

5.4.3.1.2 Call_Read_Mvb_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 10							
bus_id								reserved1							

```

Call_Read_Mvb_Status ::= RECORD
{
  bus_id          UNSIGNED8 (0..15), -- link layer identifier
  reserved1       WORD8 (=0) -- reserved
}

```

5.4.3.1.3 Reply_Read_Mvb_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 10							
bus_id								reserved1							
reserved2															
device_address															
mvp_hardware_name: STRING32															
								CHARACTER8 or '00'H							
mvp_software_name: STRING32															
								CHARACTER8 or '00'H							
device_status															
reserved3								t_ignore							
lineA_errors															
lineB_errors															

[illegible]

The Device_Status is defined by the following:

Device_Status ::= RECORD

```

{
  capabilities          BITSET4          -- basic capabilities of an
                                          MVB device
                                          (all four bits 0 = class 1)

  {
    sp      (0),          -- special - Class 1 device
    ba      (1),          -- MVB Bus_Administrator
    gw      (2),          -- gateway or Train Bus node
    md      (3)           -- messages capability
  },
  class_specific        ONE_OF          -- Class1 (C1), Bus
                                          Administrator (BA) or
                                          gateway (GW)

  {
    [C1]              WORD4,          -- for class 1, = 0
    [BA]              RECORD          -- for bus administrator
    {
      act_index        UNSIGNED2      -- lowest two bits of
                                          actualisation index

      act              BOOLEAN1       -- 1 = actualised
      mas              BOOLEAN1       -- 1 = current master
    }
    [GW]              RECORD          -- for a gateway which is not
                                          bus administrator at the
                                          same time.

    {
      std              BOOLEAN1       -- 1 = static disturbance
      dyd              BOOLEAN1       -- 1 = dynamic disturbance
      rsv1             BOOLEAN1       -- reserved, =0
      rsv2             BOOLEAN1       -- reserved, =0
    }
  }
  common_flags          BITSET8

  {
    lat      (0)         -- unused = 0
    old      (1)         -- observed line disturbed
    ssd      (2)         -- Some System Disturbance
                          (fault in the controlled
                          process, e.g. power loss).
    sdd      (3)         -- Some Disturbance of the
                          Device (device malfunction,
                          e.g. error in checksum)
    erd      (4)         -- extended reply delay enabled
    frc      (5)         -- forced device (port forced
                          in any Traffic_Store of this
                          device)
    dnr      (6)         -- Device Not Ready (e.g. not
                          initialised)
    ser      (7)         -- Reserved for Service (this
                          Station is currently
                          reserved by a Manager).
  }
}

```

NOTE The MVB_Status is defined in 3.8.4.3, which is the reference for the definition of the terms in case of discrepancy.


```

sla          (4)          -- (sla = slb = 0: no action)
                           trust Line_A, even if
                           disturbed;
slb          (5)          -- (sla = slb = 1: normal
                           redundancy)
                           trust Line_B, even if
                           disturbed
cla          (6)          -- (cla = clb = 0: no action)
                           reset frame error counter of
                           Line_A.
clb          (7)          -- (cla = clb = 1: reset both
                           counters)
                           reset frame error counter of
                           Line_B;
    }
}

```

NOTE The MVB_Control is defined in 3.8.4.4, which is the reference in case of discrepancy.

5.4.3.2.3 Reply_Write_Mvb_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 11							

```

Reply_Write_Mvb_Control ::= RECORD
{
    -- no parameters
}

```

5.4.3.3 Read_MVB_Devices

5.4.3.3.1 Description

Retrieves the list of devices which the Bus_Administrator found on the bus, together with their Device_Status.

5.4.3.3.2 Call_Read_Mvb_Devices

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 12							
bus_id								reserved1							

```

Call_Read_Mvb_Devices ::= RECORD
{
    bus_id          UNSIGNED8 (0..15),  -- identifier of the link
    reserved1       WORD8 (=0)         -- reserved
}

```

5.4.3.3.3 Reply_Read_Mvb_Devices

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 12							
bus_id								reserved1, =0							
device_address															
nr_devices															
devices_list: ARRAY [nr_devices] OF															
device_address (n)															
device_status (n)															

Reply_Read_Mvb_Devices ::= RECORD

```

{
  bus_id          UNSIGNED8 (0..15),  -- identifier of the link
  reserved1       WORD8 (=0),         -- reserved
  device_address  WORD16,              -- Device_Address device on
                                         this bus
  nr_devices      UNSIGNED16,         -- number of devices in this
                                         list
  devices_list    ARRAY [nr_devices] OF
  {
    device_address WORD16,             -- address of the device.
    device_status  Device_Status      -- copy of the Device Status of
                                         the device
  }
}

```

5.4.3.4 Write_MVB_Administrator**5.4.3.4.1 Description**

To allow fast access to data structures, the Bus_Administrator configuration list operates with offsets and indices. This applies especially to the four lists:

- the Known_Devices_List
- the Periodic_List
- the Bus_Administrators_List
- the Devices_List.

For instance, the Periodic_List is pointed to by an offset located at word 24. The size of the Periodic_List is implicitly given by the difference between this offset (periodic_list_offset) and the next offset (bus_administrators_list_offset) located at word 26.

The Periodic_List is itself divided into:

- the 11 Cycle Lists, which list for each Cycle the master frames (F_code and address) in the order in which they should be sent out. All F_codes are permitted. Each Cycle List is named according to its individual period, which is a 2^n multiple of the basic period, with $n = (1..10)$ as Cycle_1, Cycle_2, Cycle_4,... up to Cycle_1024.

- the five Split Lists, which define the actual split of the cycles into subcycles over the different periods and which contain the number of master frames to be sent in each basic period. A subcycle may not have more than 32 members. In practice, it is not possible to have more than 15 identifiers in a subcycle.

For efficiency, the Split List of two cycles are included into the same data structure.

EXAMPLE - Split_8_16 contains in the first octet, the Split List of Cycle_8 and, in the second, the Split List of Cycle_16. Since the Split List of Cycle_8 encompasses only 8 periods, while the Split List of Cycle_16 encompasses 16 periods, the Split List of Cycle_8 is repeated twice.

The offset of the Cycle List and of the Split List itself is not relative to the beginning of the message, but to the first word of the Periodic_List, called "periodic_list_offset".

The size of each list is given by the difference between its offset and the offset of the next list.

EXAMPLE - the size of Cycle_1 is given by the difference: Cycle_2_Offset - Cycle_1_Offset, the size of Cycle_32 is given by Cycle_64_Offset - Cycle_32_Offset.

An example of a MVB_Administrator list appears in Table 72.

NOTE In case of discrepancy between the following specification and the MVB specification in 3.8.4.6, the latter holds.

5.4.3.4.2 Call Write Mvb Administrator

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 13							
bus_id								reserved1							
checkword0															
configuration_version															
t_reply_max															
macro_cycles															
event_poll_strategy															
basic_period															
macrocycles_per_turn															
device_scan_strategy															
reserved2															
reserved3															
reserved4															
reserved5															
known_devices_list_offset															
reserved_list_offset															
periodic_list_offset															
bus_administrators_list_offset															
device_scan_list_offset															
end_list_offset															
known_devices_list [nr_devices]															
device_address (n)															
(offset_P0) cycle_lists_offsets ARRAY [11] OF															
cycle_n_offset (counted from offset_P0)															
split_lists_offsets ARRAY [5] OF															
split_m_n_offset (counted from offset_P0)															
cycle_lists [11]															
cycle_n ARRAY [cycle_size = cycle_n+1_offset - cycle_n_offset] OF															
f_code (n)				address (n)											
split_2_4 ARRAY [4] OF															
cycle_2								cycle_4							
split_8_16 ARRAY [32] OF															
cycle_8								cycle_16							
split_32_64 ARRAY [64] OF															
cycle_32								cycle_64							
split_128_256 ARRAY [256] OF															
cycle_128								cycle_256							
split_512_1024 ARRAY [1024] OF															
cycle_512								cycle_1024							
bus_administrators_list [nr_admins]															
void				device_address (n)											

```

UpperLowerCount ::= RECORD
{
    lower_count          UNSIGNED8
    -- auxiliary type for the split
    -- lists

    upper_count          UNSIGNED8,
    -- number of frames of the
    -- shorter cycle to be put into
    -- each basic period
    -- (this list appears twice, it
    -- is half as long as the
    -- upper_count list).

    -- number of frames of the
    -- longer cycle to be put into
    -- each basic period
}

Call_Write_Mvb_Administrator ::= RECORD
{
    bus_id                UNSIGNED8 (0..15),
    reserved1              WORD8
    -- identifier of the link
    -- reserved, =0
    checkword0             UNSIGNED16,
    -- checksum over this structure
    -- (user dependent)
    -- checkword0 is located at
    -- offset 0.

    actualisation_key      UNSIGNED16,
    -- version number of the
    -- configuration, as a 16-bit
    -- number. The most significant
    -- bit set to 1 indicates to
    -- transmit the least
    -- significant 15 bits in
    -- Mastership_Transfer_Response
    -- .
    -- The two least significant
    -- bits are copied to AX0 and
    -- AX1.

    t_reply_max            UNSIGNED16,
    -- maximum value of the
    -- expected reply delay for any
    -- device on the bus, in
    -- multiples of 1,0 µs;
    -- default (t_reply_max = 0) =
    -- 42,7 µs,
    -- maximum = 256 µs).

    macro_cycles           UNSIGNED16,
    -- duration of a macrocycle
    -- after which the master may
    -- release control if
    -- requested, in number of
    -- basic periods.

    event_poll_strategy    ENUM16
    {
        NOEVENTS          ('0000'H),
        HIPRONLY           ('4000'H),
        LOPRONLY           ('8000'H),
        HILOPRIO           ('C000'H)
    },
    -- no event polling
    -- high priority events only
    -- low priority events only
    -- high and low priority
    -- events.

    basic_period           UNSIGNED16,
    -- basic period in
    -- microseconds;
    -- allowed values: 0000 (no
    -- poll), 1000 µs.
}

```

```

macrocycles_per_turn  UNSIGNED16,          -- number of macrocycles during
                                              which the master retains
                                              control (a multiple of 1024
                                              ms).

device_scan_strategy  ENUM16,
{
    SCANKNOWNONLY  (0), -- scan all known devices
    SCANALL        (1) -- scan all device addresses
},
reserved2           WORD16 (=0),
reserved3           WORD16 (=0),
reserved4           WORD16 (=0),
reserved5           WORD16 (=0),
known_device_list_offset  UNSIGNED16,      -- offset (from checkword0)
                                              where the first address of
                                              the device list is located;
                                              (The list size is the
                                              difference between this
                                              offset and the
                                              reserved_list_offset.)

reserved_list_offset  UNSIGNED16,          -- equal to
                                              "periodic_list_offset" (list
                                              size = 0)

periodic_list_offset  UNSIGNED16,          -- offset (from checkword0)
                                              where the first F_code +
                                              address of the first cycle
                                              is located;
                                              (list size is the difference
                                              between this offset and
                                              bus_administrators_list_offs
                                              et.)

bus_administrators_list_offset  UNSIGNED16, -- offset (from
                                              checkword0) where the first
                                              address of the
                                              Bus_Administrators_List is
                                              located.

device_scan_list_offset  UNSIGNED16,      -- offset (from checkword0),
                                              where the device scan
                                              allocation list is located
                                              (unused)

end_list_offset         UNSIGNED16,      -- dummy offset indicating size
                                              of ScanList and pointing to
                                              the next location after the
                                              last one.

known_devices_list      ARRAY [nr_devices] OF -- list of the known
{                               devices
    device_address      WORD16
},
cycle_lists_offsets     RECORD          -- (nr_devices =
{                               known_devices_list_offset -
    cycle_1_offset      UNSIGNED16      reserved_list_offset)
    cycle_2_offset      UNSIGNED16      -- Device_Address of the device
    cycle_4_offset      UNSIGNED16
    cycle_8_offset      UNSIGNED16
    cycle_16_offset     UNSIGNED16
}

```

-- the periodic list begins here

-- comprising for each one of the

-- possible 11 cycles: offset

-- (counted from cycle_lists_indices)

-- where the first address of the cycle

-- is located

```

cycle_32_offset    UNSIGNED16
cycle_64_offset    UNSIGNED16
cycle_128_offset   UNSIGNED16
cycle_256_offset   UNSIGNED16
cycle_512_offset   UNSIGNED16
cycle_1024_offset  UNSIGNED16
}
split_list_offsets RECORD          -- offset (counted from
{                                  -- cycle_lists_indices) where
                                  -- the
                                  -- first address of the table
                                  -- Split_n_m
                                  -- is located
    split_2_4_offset    UNSIGNED16
    split_8_16_offset   UNSIGNED16
    split_32_64_offset  UNSIGNED16
    split_128_256_offset UNSIGNED16
    split_512_1024_offset UNSIGNED16
}
cycle_lists        RECORD          -- eleven lists, one for each
                                  -- cycle
                                  -- cycle_size = offset between
                                  -- this list and the next

cycle_1            ARRAY[cycle_size] OF FcodeAdr
cycle_2            ARRAY[cycle_size] OF FcodeAdr
cycle_4            ARRAY[cycle_size] OF FcodeAdr
cycle_8            ARRAY[cycle_size] OF FcodeAdr
cycle_16           ARRAY[cycle_size] OF FcodeAdr
cycle_32           ARRAY[cycle_size] OF FcodeAdr
cycle_64           ARRAY[cycle_size] OF FcodeAdr
cycle_128          ARRAY[cycle_size] OF FcodeAdr
cycle_256          ARRAY[cycle_size] OF FcodeAdr
cycle_512          ARRAY[cycle_size] OF FcodeAdr
cycle_1024         ARRAY[cycle_size] OF FcodeAdr
}
split_lists        RECORD
{
    split_2_4        ARRAY [ 4] OF UpperLowerCount -- see this Type
                                                    definition.
    split_8_16        ARRAY [16] OF UpperLowerCount
    split_32_64        ARRAY [64] OF UpperLowerCount
    split_128_256      ARRAY [256] OF UpperLowerCount
    split_512_1024     ARRAY [1024] OF UpperLowerCount
}
bus_administrators_list ARRAY [nr_admins] OF -- list of the known
{
    -- Bus Administrators, with:
    -- (nr_admins =
    -- device_scan_list_offset -
    -- bus_administrators_list_offs
    -- et)

    f_code            UNSIGNED4          -- reserved, = '0100'B
    device_address     UNSIGNED12        -- 12-bit address of the device
}
}

```


Table 72 - Example of mvb_administrator_list

Offset	Value	Name	Contents
000	D4EC		checksum (unused)
002	0001		actualisation_key (the two LSBs form the Actualisation_Index)
004	07D0		T_reply_max = 2000,0 µs
006	0000		macro_cycle (unused, means 1024 ms)
008	C000		event poll strategy = high and low priority events
00A	03E8		basic_period = 1 ms
00C	1000		turn duration (4096 x 1 ms)
00E	0001		device scan strategy: 1= all devices
010	0000		reserved
012	0000		reserved
014	0000		reserved
016	0000		reserved
018	0024		offset to Known_Devices_List ('024'H = 36)
01A	0036		unused (Known_Devices_List size = '036'H - '024'H = 9 words)
01C	0036		offset to Periodic_List (size = 'B36'H - '036'H = 1664)
01E	0B36		offset to Bus_Administrators_List
020	0040		offset to Unused_List (void)
022	0040		offset to End
024	0020	Known_Devices_List	device '002'H (first of a list of 9 devices)
026	0022		device '004'H bus 0
028	0024		device '024'H bus 0
02A	0028		device '028'H bus 0
02C	002C		device '02C'H bus 0
02E	0034		device '034'H bus 0
030	0036		device '036'H bus 0
032	00C8		device '0C8'H bus 0
034	00DE		device '0DE'H (last device)
036	0020	Periodic_List	Cycle_1_offset (location = '36'H + '20'H = '56'H, size = 0)
038	0020		Cycle_2_offset (location = '056'H, size = 0)
03A	0020		Cycle_4_offset (location = '056'H, size = 0)
03C	0020		Cycle_8_offset (location = '056'H, size = 0)
03E	0020		Cycle_16_offset (location = '056'H, size = '024'H)
040	0044		Cycle_32_offset (location = '7A'H, size = '0A'H)
042	004E		Cycle_64_offset
044	0052		Cycle_128_offset
046	0056		Cycle_256_offset
048	0056		Cycle_512_offset
04A	0058		Cycle_1024_offset
04C	0058	Split_List_indices:	Split_2_4_offset (location = '36'H + '58'H = '8E'H)
04E	0060		Split_8_16_offset (location = '36'H + '60'H = '96'H)
050	0080		Split_32_64_offset (location = '36'H + '80'H = 'B6'H)
052	0100		Split_128_256_offset (location = '36'H + '100'H = '136'H)
054	0300		Split_512_1024_offset

Table 72 (continued)

Offset	Value	Name	Contents
056	00C8	Cycle_16	1st F_code + address ('56'H = '36'H + '20'H)
058	00C9	(Cycle_1, Cycle_2,	2nd F_code + address
05A	03E8	Cycle_4, Cycle_8,	3rd F_code + address
05C	03EA	Cycle_1024 are void)	4th F_code + address
05E	0450		5th F_code + address
060	0452		6th F_code + address
062	0454		7th F_code + address
064	0456		8th F_code + address
066	0458		9th F_code + address
068	045A		10th F_code + address
06A	045C		11th F_code + address
06C	045E		12th F_code + address
06E	1464		13th F_code + address
070	1466		14th F_code + address
072	1468		15th F_code + address
074	146A		16th F_code + address
076	3460		17th F_code + address
078	3462		18th F_code + address (and last)
07A	015F	Cycle_32	1st F_code + address ('56'H = '36'H + '44'H)
07C	01F1		2nd F_code + address
07E	0321		3rd F_code + address
080	415E		4th F_code + address
082	4320		5th F_code + address (and last)
084	019D	Cycle_64	1st F_code + address ('84'H = '36'H + '4E'H)
086	11C2		2nd F_code + address (and last)
088	40F9	Cycle_128	1st F_code + address of Cycle_128
08A	418F		2nd F_code + address of Cycle_128 (and last)
08C	20FE	Cycle_512	1st F_code + address of Cycle_512 (and only)
08E	0000	Split_2_4	unused (void cycles)
090	0000	(4 words)	unused (void cycles)
092	0000		unused (void cycles)
094	0000		unused (void cycles)
096	0600	Split_8_16	1st period: 6 of Cycle_16, 0 of Cycle_8 (void)
098	0600	(16 words)	2nd period: 6 of Cycle_16, 0 of Cycle_8 (void)
09A	0400		3rd period: 4 of Cycle_16, 0 of Cycle_8 (void)
09C	0200		4th period: 6 of Cycle_16, 0 of Cycle_8 (void)
...	...		
0B4	0000		16th period: 0 of Cycle_16, 0 of Cycle_8 (void)
0B6	0000	Split_32_64	1st period: 0
0B8	0000	(64 words)	2nd period: 0
0BA	0001		3rd period: 1 of Cycle_32
0BC	0002		4th period: 2 of Cycle_32
0BE	0201		5th period: 2 of Cycle_64, 1 of Cycle_32
0C0	0001		6th period: 1 of Cycle_32
0C2	0000		7th period: 0 of Cycle_64, 0 of Cycle_32
...	...		...

Table 72 (concluded)

Offset	Value	Name	Contents
134	0000	Split_128_256 (256 words)	64th period: 0
136	0000		1st period: 0
			2nd period
...	...		...
334	0000	Split_512_1024 (1024 words)	256th period:
336	0000		1st period
338	0000		2nd period
...	...		...
340	0001		6th period: 1 of Cycle_512
...	...		...
B34	0000		1024th period
B36	0002	Bus_Admistrators_List	first device in Bus_Admistrators_List
B38	0004		2nd device in Bus_Admistrators_List
(B3A)		(Unused List, End)	dummy

5.4.3.4.3 Reply_Write_Mvb_Administrator

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 13							

```
Reply_Write_Mvb_Administrator ::= RECORD
{ }
```

```
-- no parameters
```

5.4.4 WTB link services**5.4.4.1 Read_Wtb_Status****5.4.4.1.1 Description**

The service reads the status of a WTB link layer of a node. If this Station has no Train Bus, it returns an error and the Reply_Message consists only of the header.

The returned data corresponds to the data structures WTB_Status and WTB_LLStatisticData in Clause 4.

NOTE In case of discrepancy, the parameter definitions in Clause 4 (WTB) hold.

5.4.4.1.2 Call_Read_Wtb_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 20							
bus_id								reserved1							

```
Call_Read_Wtb_Status ::= RECORD
```

```
{
  bus_id          UNSIGNED8 (0..15)  -- identifier of the link
  reserved1       WORD8 (=0)         -- reserved
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 20							
bus_id								node_address							
wtb_hardware_id								wtb_software_id							
hardware_state								link_layer_state							
net_inhibit								node_address							
node_orient								node_strength							
node_frame_size								node_period							
node_type								node_version							
node_report								user_report							
basic_period_count															
.....															
inauguration_count															
topography_count															
transmitted_md_count															
received_md_count															
line_status_a1															
(transmitted_count)															
(received_count)															
(errors_count)															
(timeouts_count)															
line_status_a2															
line_status_b1															
line_status_b2															
line_switch_count															

```
Line_Status ::= RECORD
```

```
{
transmitted_count    UNSIGNED32,    -- incremented for each sent
                                frame
received_count       UNSIGNED32,    -- incremented for each
                                received frame
frame_errors_count   UNSIGNED16,    -- incremented for each
                                erroneous frame
frame_timeouts_count UNSIGNED16,    -- incremented for each timed-
                                out frame
}
```

Reply_Read_Wtb_Status ::=

RECORD

```
{
bus_id                UNSIGNED8 (0..15)    -- identifier of the link
node_address          WORD8              -- Node_Address (127 if unnamed)
wtb_hardware_id       UNSIGNED8          -- identifier of the hardware
wtb_software_id       UNSIGNED8          -- identifier of version of the link layer software.
hardware_state        ENUM8
{
LS_OK                 (0),               -- WTB_LS_OK      correct operation
LS_FAIL               (1),               -- WTB_LS_FAIL   hardware failure
},
link_layer_state      Ls_T_State,         -- major state in which the node is currently
net_inhibit           ENUM8              -- 1: some node inhibits inauguration
node_address          UNSIGNED8          -- assigned node address
node_orient           ENUM8              -- orientation of the node relative to the master node
{
L_UNKNOWN             (0),               -- WTB_LS_UNKNOWN
L_SAME                (1),               -- WTB_LS_SAME
L_INVERSE             (2),               -- WTB_LS_INVERSE
},
node_strength         ENUM8              -- strength of the node
{
L_UNDEFINED           (0)                -- node strength undefined
L_SLAVE               (1)                -- node is slave only
L_STRONG              (2)                -- node is strong master
L_WEAK                (3)                -- node is weak master
},
node_frame_size       UNSIGNED8,         -- size of the process data frame sent by the node.
node_period           UNSIGNED8,         -- desired individual period in multiples of T_bp
node_type             UNSIGNED8,         -- descriptor of node abilities (part of Node_Key)
node_version          UNSIGNED8,         -- descriptor of node abilities (part of Node_Key)
node_report           Node_Report,       -- 8-bit node report, as expressed in Status Response
```

```

user_report          User_Report,      -- 8-bit user report, as
                                -- expressed in Status_Response
                                -- the following corresponds to
                                -- Type_LLStatisticData
basic_period_count   UNSIGNED32        -- incremented for each
                                -- Basic_Period
inauguration_count   UNSIGNED16        -- incremented for each
                                -- inauguration
topography_count     UNSIGNED16        -- incremented for each new
                                -- topography
transmitted_md_count UNSIGNED32,        -- incremented for each sent
                                -- Message_Data_Response
received_md_count    UNSIGNED32,        -- incremented for each
                                -- received
                                -- Message_Data_Response
line_status_a1       Type_LineStatus,  -- statistics for A1, see type
                                -- definition
line_status_a2       Type_LineStatus,  -- statistics for A2, see type
                                -- definition
line_status_b1       Type_LineStatus,  -- statistics for B1, see type
                                -- definition
line_status_b2       Type_LineStatus,  -- statistics for B2, see type
                                -- definition
line_switch_count    UNSIGNED32        -- incremented for each line
                                -- switch
}

```

5.4.4.2 Write_Wtb_Control

5.4.4.2.1 Description

Sets parameters in the Train Bus node.

5.4.4.2.2 Call_Write_Wtb_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 21							
bus_id								reserved1							
rsv1	rsv2	rsv3	rsv4	rsv5	cs1	slp	alw	inh	rmv	snm	stm	wkm	slv	cnf	rst

Call_Write_Wtb_Control ::= RECORD

```

{
  bus_id      UNSIGNED8 (0..15),  -- identifier of the link
  reserved1   WORD8 (=0)         -- reserved
  command     BITSET16
  {
    rsv1      (0),               -- reserved
    rsv2      (1),               -- reserved
    rsv3      (2),               -- reserved
    rsv4      (3),               -- reserved
    rsv5      (4),               -- reserved
    cs1       (5),               -- cancel sleep
    slp       (6),               -- set sleep
    alw       (7),               -- allow
    inh       (8),               -- inhibit
    rmv       (9),               -- remove
    snm       (10),              -- start naming
    stm       (11),              -- set strong master
    wkm       (12),              -- set weak master

```

```

    slv      (13),      -- set slave
    cnf      (14),      -- configure
    rst      (15)       -- reset node
}
}

```

5.4.4.2.3 Reply_Write_Wtb_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 21							
bus_id								reserved1							

```
Reply_Write_Wtb_Control ::= RECORD
```

```
{
bus_id          UNSIGNED8 (0..15),  -- identifier of the link
reserved1      WORD8 (=0)          -- reserved
}
```

5.4.4.3 Read_Wtb_Nodes

5.4.4.3.1 Description

Retrieves the list of nodes which the master found on the bus, together with their Node Status Word (this command is always sent to node 01).

5.4.4.3.2 Call_Read_Wtb_Nodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 22							
bus_id								reserved1 = 0							

~~Call_Read_Wtb_Nodes ::= RECORD~~

```
{
bus_id          UNSIGNED8 (0..15)    -- identifier of the link
reserved1      WORD8 (=0)           -- reserved
}
```

5.4.4.3.3 Reply_Read_Wtb_Nodes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 22							
bus_id								node_address							
reserved1								nr_nodes							
bottom_node								top_node							
node_status_list: ARRAY [MAX_NODES] OF															
node report								user report							

```

Reply_Read_Wtb_Nodes ::= RECORD
{
  bus_id            UNSIGNED8 (0..15),  -- identifier of the link
  node_address      UNSIGNED8,          -- address of this node (127 if
                                         unnamed)
  reserved1         WORD8 (=0),         -- reserved
  nr_nodes          UNSIGNED8,          -- number of nodes in the
                                         composition.
  bottom_node       UNSIGNED8,          -- Node_Address of the node
                                         with the lowest address.
  top_node          UNSIGNED8,          -- Node_Address of the node
                                         with the highest address.
  node_status_list  ARRAY[MAX_NODES] OF
  {
                                         -- list of node status,
                                         beginning with bottom node,
                                         in the order in which nodes
                                         are located, and ending
                                         with top node, consisting
                                         of:
    node_report      Node_Report        -- Node_Report of the different
                                         nodes
    user_report      User_Report        -- User_Report of the different
                                         nodes
  }
}

```

5.4.4.4 Read_Wtb_Topography

5.4.4.4.1 Description

Reads the Topography. The format of the Topography is specified in the parameter list, since there are small differences between the Topography information as transmitted over the WTB, as available at the WTB supervisory interface, and as available through network management.

5.4.4.4.2 Call_Read_Wtb_Topography

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 24							
bus_id								reserved1							

```

Call_Read_Wtb_Topography ::= RECORD
{
  bus_id            UNSIGNED8 (0..15),  -- identifier of the link
  reserved1         WORD8 (=0)         -- reserved
}

```


5.4.5 Variables services

5.4.5.1.1 Description

5.4.5.1.2 Call Read Ports Configuration

```
Call_Read_Ports_Configuration ::= RECORD
```

5.4.5.1.3 ~~Reply_Read_Ports_Configuration~~

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 30							
nr_ports															
ports_list: ARRAY [nr_ports] OF															
bus_id				port_address											
f code				src	snk	twc	frc	port size							

5.4.5.2.3 Reply_Write_Ports_Configuration

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 31							
bus_id								reserved1							

```

Reply_Write_Ports_Configuration ::= RECORD
{
    bus_id                UNSIGNED8 (0..15)    -- traffic store id as seen by
                                                the station
    reserved1             WORD8 (=0)           -- reserved
}
    
```

5.4.5.3 Read_Variables

5.4.5.3.1 Description

Reads the value, check variable and freshness of a cluster of variables.

Each variable is identified by its position and that of its check variable within the Traffic_Store and its type and size.

Variables may be fetched individually, by PV_Sets or by PV_Clusters.

If consecutive variables belong to the same dataset, access shall be consistent (by PV_Set)

The Agent shall respond with the list of values in the same order as the variables have been listed in the Call_Message.

Variable values shall be transported in management messages in the same format as they would be transmitted as process data over the bus.

Each variable value shall begin at a new word boundary.

Variables which do not fill a 16-bit word shall be right-justified and, if they are signed, be sign-extended.

Variables larger than 16 bits, but whose size is not a multiple of 16 bits, shall be right-justified and padded by "0" to the next word16 boundary.

EXAMPLES

1 - An 8-bit integer with the value '0111 1111'B (+127) is transmitted as '0000 0000 0111 1111'B.

2 - An 8-bit integer with the value '1111 1111'B (-1) is transmitted as '1111 1111 1111 1111'B.


```

Reply_Read_Variables ::=                                RECORD
{
    nr_vars                UNSIGNED16,                -- number of variables in the
                                                         list.
    values_list            ARRAY [nr_vars] OF
    {
        value              ANY,                      -- for each variable in the
                                                         list and in same order):
                                                         -- value with a format given by
                                                         Size and Type of the
                                                         PV_NAME, aligned to a 16-
                                                         word boundary.
                                                         CHARACTER8 is an exception,
                                                         as it is interpreted as a
                                                         special case of ARRAY [0..0]
                                                         OF CHARACTER8: the
                                                         character occupies the high-
                                                         order octet, the low-order
                                                         octet being occupied by
                                                         '00'h.
    }
    check_var              ANTIVALENT2,                -- associated check variable,
                                                         or '11'B if none is
                                                         specified.
    freshness              UNSIGNED14                 -- value of freshness of
                                                         variable in milliseconds
                                                         (maximum: 4096 ms).
}
}

```

NOTE A Check_Variable may also be transported as any other (ANTIVALENT2) variable.

5.4.5.4 Write_Force_Variables

5.4.5.4.1 Description

Forces a cluster of variables to a defined value.

Forcing a variable shall set the "frc" bit in the Device_Status of the corresponding MVB Traffic_Store and in the Station_Status.

The check field shall be set to the value specified by the check variable.

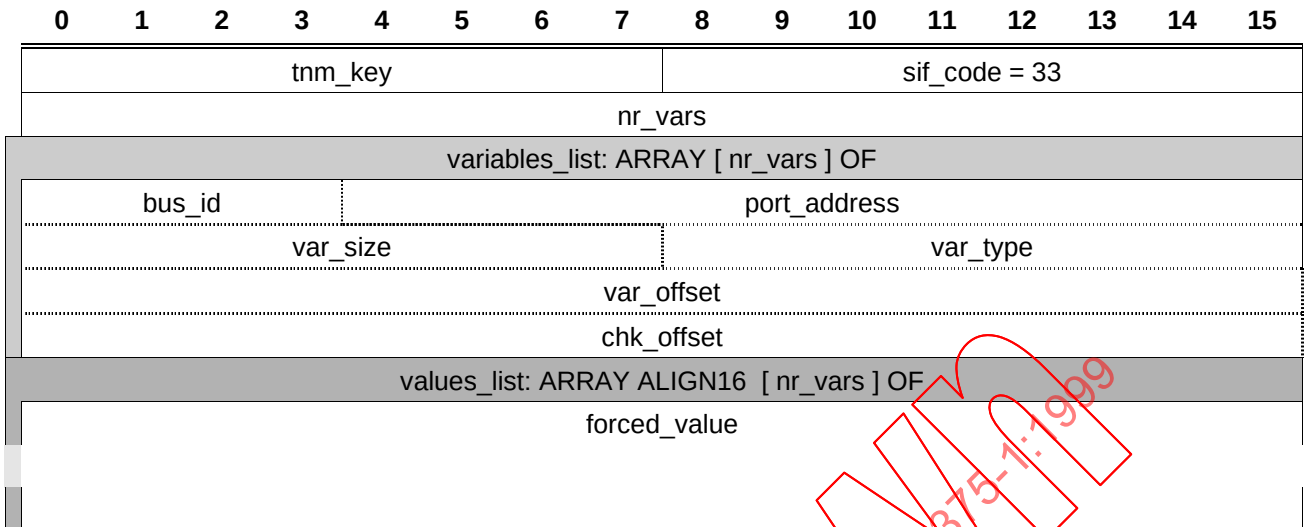
Each variable value shall begin at a new word boundary.

Variables which do not fill a 16-bit word shall be right-justified and, if they are signed, be sign-extended.

Variables larger than 16 bits, but whose size is not a multiple of 16 bits, shall be right-justified and padded by "0" to the next word16 boundary.

NOTE There is no "frc" bit in the Train Bus status, but this information can be deduced from the Station_Status.

5.4.5.4.2 Call_Write_Force_Variables

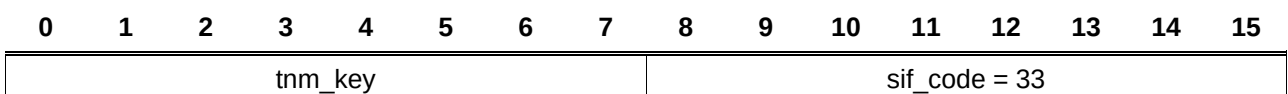


```

Call_Write_Force_Variables ::= RECORD
{
  nr_vars          UNSIGNED16,          -- number of variables in the
                                         list
  variables_list   ARRAY [ nr_vars ] OF -- one entry for each variable
  {
    bus_id         UNSIGNED4,          -- identifier of traffic store
    port_address   WORD12,            -- port address in the traffic
                                         store
    var_size       UNSIGNED8,          -- size in bits (simple types)
                                         or elements (structured
                                         types) according to 2.2
                                         (RTP)
    var_type       UNSIGNED8,          -- type of the variable
                                         according to 2.2 (RTP)
    var_offset     UNSIGNED16,         -- offset of the variable
    chk_offset     UNSIGNED16         -- offset of the check variable
                                         ('FFFF'H if unused).
  },
  values_list      ARRAY ALIGN16 [nr_vars] OF
  {
    -- list of values to force, in
    -- same order as
    -- variables_list. (same rules
    -- as for read variables apply)
    forced_value   ANY                -- value of the variable in the
                                         format given by the PV_Name,
                                         as it would be sent as
                                         process data on the bus.
  }
}

```

5.4.5.4.3 Reply_Write_Force_Variables



```

Reply_Write_Force_Variables ::= RECORD
{
}
-- no parameters

```

5.4.5.5 Write_Unforce_Variables

5.4.5.5.1 Description

Unforces all listed variables. This service, if successful, shall reset the "frc" bit in the status of the corresponding link layer and reset the 'frc' bit in the Station_Status if all variables of this station have been unforced.

5.4.5.5.2 Call_Write_Unforce_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 35							
nr_vars															
variables_list: ARRAY[nr_vars] OF															
bus_id				port_address											
var_size								var_type							
var_offset															
chk_offset															

```

Call_Write_Unforce_Variables ::= RECORD
{
  nr_vars          UNSIGNED8,      -- number of variables in the
                                   list.
  variables_list   ARRAY [ nr_vars ] OF
  {
    bus_id         UNSIGNED4,      -- identifier of traffic store
    port_address   WORD12,        -- port address in the traffic
                                   store
    var_size       UNSIGNED8,      -- size in bits (simple types)
                                   or number of elements
                                   (structured types) according
                                   to 2.2 (RTP)
    var_type       UNSIGNED8,      -- type of the variable
                                   according to 2.2 (RTP)
    var_offset     UNSIGNED16,     -- offset of the variable
    chk_offset     UNSIGNED16     -- offset of the check variable
                                   ('FFFF'H if unused).
  }
}

```

5.4.5.5.3 Reply_Write_Unforce_Variables

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 35							

```

Reply_Write_Unforce_Variables ::= RECORD
{ }
-- no parameters

```

5.4.5.6 Write_Unforce_All

5.4.5.6.1 Description

Unforces all variables in a particular Traffic_Store.

This service, if successful, clears the "frc" bit in the Device_Status corresponding to that Traffic_Store and in the Station_Status, once all Traffic_Stores are unforced.

5.4.5.6.2 Call_Write_Unforce_All

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 37							
ts0								link_set							
								ts15							

Call_Write_Unforce_All ::= RECORD

```
{
  link_set          Link_Set          -- bitset indicating the
                                         Traffic_Stores to be
                                         unforced, Traffic_Store 0
                                         being represented by the bit
                                         with offset 0
}
```

5.4.5.6.3 Reply_Write_Unforce_All

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 37							

Reply_Write_Unforce_All ::= RECORD

```
{
  } -- no parameters
```

5.4.5.7 Read_Variable_Bindings

5.4.5.7.1 Description

Reads the binding of all variables supported by a station. The order is arbitrary.

NOTE This service supports the marshalling of internal device variables with network variables.

5.4.5.7.2 Call_Read_Variable_Bindings

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 38							

Call_Read_Variable_Bindings ::= RECORD

```
{
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 38							
nr_vars															
variables_list: ARRAY[nr_vars] OF															
variable_name: STRING32															
(CHARACTER8)								CHARACTER8 or '00'H							
var_properties								individual_period							
standard_type															
bus_id				port_address											
var_size								var_type							
var_offset															
chk_offset															

[illegible]

5.4.6.1.3 Reply_Read_Messenger_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 40							
reserved1															
messenger_name: STRING32															
								CHARACTER8 or '00'H							
send_time_out								alive_time_out							
ack_time_out								credit							
reserved2								packet_size							
instances								multicast_window							
messages_sent															
messages_received															
packets_sent															
packet_retries															
multicast_retries															

Reply_Read_Messenger_Status ::= RECORD

```

{
  reserved1          WORD16 (=0),          -- reserved
  messenger_name     STRING32,             -- version of the messenger
                                              software, preferably in
                                              format:
                                              xxxx-Vz.z-dd.mm.yy
  send_time_out      UNSIGNED8,            -- time-out after which
                                              producer retries, in
                                              multiples of 64 ms
  alive_time_out     UNSIGNED8,            -- time-out after which the
                                              consumer disconnects, in
                                              seconds
  ack_time_out       UNSIGNED8,            -- time-out after which replier
                                              acknowledges all received
                                              data packets, in multiples
                                              of 64 ms
  credit             UNSIGNED8,            -- number of data packets the
                                              producer may send before it
                                              receives an acknowledgement
                                              for any of them
  reserved2          WORD8 (=0),           -- reserved
  packet_size        UNSIGNED8,            -- size of the packet in octets
  instances          UNSIGNED8,            -- number of supported
                                              instances for each replier.
  multicast_window   UNSIGNED8,            -- window size for the
                                              multicast mechanism. If 0,
                                              no multicast is supported.
  messages_sent      UNSIGNED32,            -- counter (with wrap-around)
                                              counting the number of
                                              messages sent by this
                                              Station
  messages_received  UNSIGNED32,            -- counter (with wrap-around)
                                              counting the number of
                                              messages received on this
                                              Station
  packets_sent       UNSIGNED32,            -- counter (with wrap-around)
                                              counting the number of
                                              packets sent on this
                                              Station
  packet_retries     UNSIGNED32,            -- counter (with wrap-around)
                                              counting the number of
                                              retried packets in the
                                              single-cast protocol
  multicast_retries   UNSIGNED32            -- counter (with wrap-around)
                                              counting the number of
                                              retried packets in the
                                              multicast protocol
}

```

5.4.6.2 Write_Messenger_Control

5.4.6.2.1 Description

Sets parameters in the messenger.

5.4.6.2.2 Call_Write_Messenger_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 41							
reserved1															
messenger_name: STRING32															
CHARACTER8								CHARACTER8 or '00'H							
send_time_out								alive_time_out							
ack_time_out								credit							
reserved2								packet_size							
rsv1	rsv2	rsv3	mcr	pkp	pks	mgr	mgs	multicast_window							

```

Call_Write_Messenger_Control ::= RECORD
{
    reserved1                WORD16 (=0),           -- reserved
    messenger_name            STRING32,              -- version of the messenger
                                                              software, preferably in
                                                              format:
                                                              xxxx-V2.z-dd.mm.yy
    send_time_out             UNSIGNED8,             -- time-out after which the
                                                              producer retries, expressed
                                                              in multiples of 64 ms
    alive_time_out            UNSIGNED8,             -- time-out after which
                                                              consumer disconnects in
                                                              seconds
    ack_time_out              UNSIGNED8,             -- time-out after which the
                                                              replier acknowledges all
                                                              received data packets, in
                                                              multiples of 64 ms
    credit                    UNSIGNED8,             -- number of data packets which
                                                              may be sent before an
                                                              acknowledgement is received
    reserved2                 WORD8 (=0),           -- reserved
    packet_size               UNSIGNED8,             -- size of the packet, in
                                                              octets
    clear_counter             BITSET8               -- clear the following
                                                              counters:
        {
            rsv1,                -- reserved
            rsv2,                -- reserved
            rsv3,                -- reserved
            mcr,                 -- multicast retries counter
            pkr,                 -- packet retries counter
            pks,                 -- sent packets counter
            mgr,                 -- received messages counter
            mgs,                 -- sent messages counter
        },
    multicast_window          UNSIGNED8             -- window size for the
                                                              multicast mechanism.
}
-- If 0, no multicast is supported.

```


5.4.6.2.3 Reply_Write_Messenger_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 41							

```

Reply_Write_Messenger_Control ::= RECORD
  {}
  -- no parameters

```

5.4.6.3 Read_Function_Directory**5.4.6.3.1 Description**

Reads the Function_Directory at a Station.

5.4.6.3.2 Call_Read_Function_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 42							

```

Call_Read_Function_Directory ::= RECORD
  {}
  -- no parameters

```

5.4.6.3.3 Reply_Read_Function_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 42							
reserved1								nr_functions							
function_list: ARRAY[nr_functions] OF															
function_id								station_id							

```

Reply_Read_Function_Directory ::= RECORD
{
  reserved1      WORD8 (=0)      -- reserved
  nr_functions   UNSIGNED8,      -- number of entries in the
                                list
  function_list  ARRAY[nr_functions] OF
  {
    function_id  UNSIGNED8,      -- Function_Identifier
    station_id   UNSIGNED8      -- corresponding
                                Station_Identifier
  }
}

```

5.4.6.4 Write_Function_Directory**5.4.6.4.1 Description**

Writes the Function_Directory at a Station.

5.4.6.4.2 Call_Write_Function_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 43							
clear_directory								nr_functions							
function_list: ARRAY [nr_functions] OF															
function_id								station_id							

```

Call_Write_Function_Directory ::= RECORD
{
  clear_directory      ENUM8
  {
    REPLACE      (0),          -- do not clear, just replace
                                -- entries
    CLEARFIRST   (1)          -- clear the directory before
                                -- inserting
  },
  nr_functions        UNSIGNED8, -- number of entries in the
                                -- list
  function_list       ARRAY [nr_functions] OF
  {
    function_id       UNSIGNED8, -- Function_Identifier
    station_id        UNSIGNED8  -- corresponding
                                -- Station_Identifier
  }
}

```

5.4.6.4.3 Reply_Write_Function_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 43							

```

Reply_Write_Function_Directory ::= RECORD
{
}
-- no parameters

```

5.4.6.5 Read_Station_Directory

5.4.6.5.1 Description

Reads the Station_Directory at a Station (if it exists)

5.4.6.5.2 Call_Read_Station_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 44							

```

Call_Read_Station_Directory ::= RECORD
{
}
-- no parameters

```



```

Call_Write_Station_Directory ::= RECORD
{
  clear_directory      ENUM8
  {
    REPLACE      (0),          -- do not clear, just replace
                                entries
    CLEARFIRST   (1)          -- clear the directory before
                                inserting
  },
  nr_stations          UNSIGNED8,  -- number of stations in the
                                list
  station_list         ARRAY [nr_stations] OF
  {
    station_id        UNSIGNED8,  -- Station Identifier
    next_station_id    UNSIGNED8,  -- Station Identifier of
                                Next_Station
    bus_id             UNSIGNED8 (0..15), -- identifier of the link where
                                Next_Station is
    reserved1          WORD8 (=0),  -- reserved
    device_address     UNSIGNED16   -- address of the device which
                                carries Next_Station
  }
}

```

5.4.6.6.3 Reply_Write_Station_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 45							

```

Reply_Write_Station_Directory ::= RECORD
{
}
-- no parameters

```

5.4.6.7 Read_Group_Directory

5.4.6.7.1 Description

Reads the Group_Directory at a Station (if it exists)

5.4.6.7.2 Call_Read_Group_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 46							

```

Call_Read_Group_Directory ::= RECORD
{
}
-- no parameters

```

5.4.6.7.3 Reply_Read_Group_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 46							
g0	g1	g2	g3	g4	g5	g6	g7	g8	g9	g10	g11	g12	g13	g14	g15
g16															g31
g32															g47
g48															g63

```

Reply_Read_Group_Directory ::= RECORD
{
  group_list          BITSET64          -- one bit set for each of the
                                         possible 64 groups to which
                                         a station may belong, offset
                                         0 being group 0.
}

```

5.4.6.8 Write_Group_Directory**5.4.6.8.1 Description**

Writes the Group_Directory at a Station (if it exists).

5.4.6.8.2 Call_Write_Group_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 47							
g0	g1	g2	g3	g4	g5	g6	g7	g8	g9	g10	g11	g12	g13	g14	g15
g16															g31
g32															g47
g48															g63

```

Call_Write_Group_Directory ::= RECORD
{
  group_list          BITSET64          -- one bit for each of the
                                         possible 64 groups a Station
                                         may belong to, bit 0
                                         identifying group 0
}

```

5.4.6.8.3 Reply_Write_Group_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 47							

```

Reply_Write_Group_Directory ::= RECORD
{
}
-- no parameters

```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 49							
clear_directory								nr_nodes							
nodes_list: ARRAY [nr_nodes] OF															
node_address								reserved1							
device address															

```

Call_Write_Node_Directory ::= RECORD
{
  clear_directory      ENUM8
  {
    REPLACE      (0),          -- do not clear, just replace
                                entries
    CLEARFIRST   (1)          -- clear the directory before
                                inserting
  },
  nr_nodes            UNSIGNED8,  -- number of Nodes in the list
  nodes_list          ARRAY [nr_nodes] OF
  {
    node_address      UNSIGNED8,  -- 8-bit Node_Address
    reserved1         WORD8 (=0),  -- reserved
    device_address     UNSIGNED16  -- Device_Address of the nodes
  }
}

```

5.4.6.10.3 Reply_Write_Node_Directory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 49							

```

Reply_Write_Node_Directory ::= RECORD
{
}
-- no parameters

```

5.4.7 Domain services

5.4.7.1 Read_Memory

5.4.7.1.1 Description

This service reads in one operation several memory regions, each consisting of a number of identical, consecutive items, which size is an integral multiple of one octet, each octet having an address.

The alignment shall be respected:

- If the item size is 1, the memory region may begin at an odd or an even address,
- If the item size is 2, the memory region shall begin at an even address,
- If the item size is 4, the memory region shall begin at an address divisible by 4.

5.4.7.1.2 Call_Read_Memory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 50							
reserved1								nr_regions							
region_list ARRAY [nr_regions]															
base_address															
nr_items															
reserved2								item_size							

Call_Read_Memory ::= RECORD

```

{
  reserved1          WORD8 (=0),
  nr_regions          UNSIGNED8,      -- number of individual regions
                                   -- to be read
  region_list         ARRAY[nr_regions] OF
  {
    base_address      UNSIGNED32,      -- base address of region
    nr_items          UNSIGNED16,      -- size of region, in multiples
                                   -- of the item size
    reserved2         WORD8 (=0),      -- reserved
    item_size         UNSIGNED8,      -- size of each item in octets,
                                   -- allowed values: 1,2,4.
  }
}

```

5.4.7.1.3 Reply_Read_Memory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 50							
reserved1								nr_regions							
region_values: ARRAY[nr_regions] OF															
nr_octets															
item_value_list: ARRAY ALIGN16 [nr_octets] OF WORD8															
first octet at even address (or '00'H if start odd)								second first octet if start odd							
item_value															
last octet (if start odd and region size even or start even and region size odd)								last octet (or '00'H if start odd and region size even or start even and region size odd)							

Reply_Read_Memory ::= RECORD

```

{
  reserved1          WORD8 (=0),      -- reserved
  nr_regions          UNSIGNED8,      -- number of individual regions
                                   -- actually read
  region_values       ARRAY [nr_regions] OF
  {
    -- array of region_values, each
    -- consisting of:

```



```

nr_items          UNSIGNED16,          -- size of the region in
                                         multiples of the item size
reserved2         WORD8 (=0),          -- reserved
item_size         UNSIGNED8            -- size in octets of each item,
                                         allowed values: 1,2,4.
},
region_value_list  ARRAY [nr_regions] OF
{
item_value_list    ARRAY ALIGN16 [nr_items] OF
{
ONE_OF [item_size]
{
1:                WORD8,              -- octets at even addresses are
                                         transmitted first
2:                WORD16,             -- written doublet by doublet
4:                WORD32              -- written quadlet by quadlet
}
}
}
}

```

NOTE Octets located at an even memory address are always transmitted with an even octet offset in the message. If the memory region begins at an odd address, the first transmitted element is meaningless. Conversely, if the memory region begins at an even address and the number of octets is odd, the last octet is meaningless. If the base address is odd and the number of octets is even, the first and the last octet are meaningless. The field 'nr_octets' contains the effectively transmitted number of octets, it is only identical to the region size when the start is even and the number of octets is even.

5.4.7.2.3 Reply_Write_Memory

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Inm_key								sif_code = 51							

```

Reply_Write_Memory ::= RECORD
{
-- no parameters

```

5.4.7.3 Download_Setup

5.4.7.3.1 Description

Download set-up prepares the downloading of a domain, which follows in different segments.

If an interval of more than 16 s elapses between the consecutive loading of two segments, the Agent shall reset the Station.

5.4.7.3.2 Call_Write_Download_Setup

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 53							
reserved1								download_command							
reserved2								download_time_out							
reserved3								nr_domains							
domain_list: ARRAY [nr_domains] OF															
base address															
domain_size															

Call_Write_Download_Setup ::= RECORD

```

{
  reserved1          WORD8 (=0),
  download_command   ENUM8,
  {
    DNLD_PREPARE      (0),          -- force the Station to start
                                     the download program. Other
                                     parameters are ignored.
    DNLD_CHECK_ONLY,  (1)          -- check whether download
                                     parameters are legal,
                                     (according to bank,
                                     base_address, size and
                                     partition) but without
                                     affecting the Station.
    DNLD_START_ERASE  (2),          -- if parameters are valid,
                                     invalidate domains, erase
                                     memory and prepare memory
                                     for downloading.
    DNLD_START_NOERASE (3),        -- if parameters are valid,
                                     invalidate domains, and
                                     prepare memory for
                                     downloading.
    DNLD_TERMINATE_BOOT (4),       -- terminate download and
                                     restart
    DNLD_TERMINATE_NOBOOT (5),    -- terminate download and stops
                                     time-out counter. wait for
                                     further service calls.
    DNLD_VERIFY       (6)         -- call the verify procedure
                                     for this domain.
  },
  reserved2          WORD8 (=0)
  download_time_out   UNSIGNED8,  -- time allowed between loading
                                     of two segments in seconds
                                     (maximum 16 s).

  reserved3          WORD8 (=0)
  nr_domains          UNSIGNED8,   -- number of domains to set up
  domain_list         ARRAY [nr_domains] OF
  {
    base_address      WORD32,      -- base address of domain in
                                     Agent's address space.
    domain_size        WORD32      -- size of domain in octets.
  }
}

```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 53							
max_segment_size															
reserved1								nr_domains							
setup_result_list ARRAY [nr_domains] OF ENUM8															
first setup_result								second setup_result							
last setup_result if nr_domains is odd								last setup_result if nr_domains is even, else '00'H							

```

Reply_Write_Download_Setup ::= RECORD
{
    max_segment_size      UNSIGNED32,      -- maximum size of
                                         download/upload buffer
    reserved1             WORD8 (=0),
    nr_domains            UNSIGNED8,      -- copy of nr_domains in Call
    setup_result_list     ARRAY [nr_domains] OF
        setup_result      ENUM8
        {
            DOMAIN_OK      (0),          -- domain successfully set
            DOMAIN_BAD_BASE_ADDR (1),    -- invalid domain base address
            DOMAIN_BAD_SIZE (2),          -- invalid domain size
            DOMAIN_ERASE_ERR (3),          -- domain may not be erased
            DOMAIN_WRITE_ERR (4),          -- domain may not be written
            DOMAIN_BAD_CHECKSUM (5),      -- incorrect checksum
        }
}

```

5.4.7.4.1 Description

This service transmits a segment of a defined size into the domain opened by Write Download Setup.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 55							
reserved1								domain_id							
segment_base_address															
segment_size															
segment_values: ARRAY [segment_size] OF															
first octet at even address or '00'H								second octet or first octet at odd address							
octet_element															
last or second last octet								last octet or '00'H							

```

Call_Write_Download_Segment ::= RECORD
{
  reserved1          WORD8 (=0),          -- padding
  domain_id          UNSIGNED8,          -- identifies the domain (array
                                         index of the domain in
                                         domain_list of the last
                                         Call_Write_Download_Setup)

  segment_base_address  UNSIGNED32,      -- base address of the segment
                                         (may be odd)
  segment_size         UNSIGNED32,      -- size of the segment in
                                         octets
  segment_values       ARRAY [segment_size] OF
  {
    octet_element      WORD8            -- list of octets
  }
}

```

5.4.7.4.3 Reply_Write_Download_Segment

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 55							

```

Reply_Write_Download_Segment ::= RECORD
{
}
-- no parameters

```

5.4.8 Task services

5.4.8.1 Read_Tasks_Status

5.4.8.1.1 Description

This service retrieves the name and status of the tasks installed in a Station.

5.4.8.1.2 Call_Read_Tasks_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 60							

```

Call_Read_Tasks_Status ::= RECORD
{
}
-- no parameters

```

5.4.8.1.3 Reply_Read_Tasks_Status

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 60							
reserved1								nr_tasks							
tasks_list: ARRAY [nr_tasks] OF															
task_name: STRING16															
								CHARACTER8 or '00'H							
priority								status							
cpu_load															
stack_margin															
task_comment: STRING26															
								CHARACTER8 or '00'H							

```

Reply_Read_Tasks_Status ::= RECORD
{
  reserved1          WORD8 (=0),          -- padding
  nr_tasks           UNSIGNED8,           -- number of returned task
                                           -- status descriptions
  tasks_list         ARRAY [nr_tasks] OF
  {
    task_name        STRING16,           -- task name or task number as
                                           -- a string
    priority         UNSIGNED8,          -- task priority (0 = highest
                                           -- priority)
    status           ENUM8               -- task status
    {
      READY         (0),
      SUSPENDED     (1),
      PENDING       (2),
      RUNNING       (3),
      FAULTY        (4)
    },
    cpu_load         UNSIGNED16,          -- CPU load generated by this
                                           -- task in % (0..100%);
                                           -- other values means CPU load
                                           -- measuring not supported)
    stack_margin     UNSIGNED16,          -- stack margin of this task
                                           -- ('FFFF'H = service not
                                           -- supported)
    task_comment     STRING26
  }
}

```

5.4.8.2 Write_Tasks_Control

5.4.8.2.1 Description

Stops or starts all tasks.

The bit "dnr" (Device Not Ready) shall be set in the Station_Status and in the Device_Status of all MVB link layers when stop is requested, and cleared after a successful start.

5.4.8.2.2 Call_Write_Tasks_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 61							
command								task_id							

Call_Write_Tasks_Control ::= RECORD

```

{
  command          ENUM8
  {
    STOP_TASK      (0)          -- stop all tasks
    START_TASK     (1)          -- start all tasks
  }
  task_id          UNSIGNED8    -- identifies the task (array
                                index of the task in
                                tasks_list of
                                Reply_Read_Tasks_Status) to
                                start or stop, 'FF' H
                                starts/stops all tasks
}

```

5.4.8.2.3 Reply_Write_Tasks_Control

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 61							

Reply_Write_Tasks_Control ::= RECORD

```

{}
-- no parameters

```

5.4.9 Clock services**5.4.9.1 Read_Clock****5.4.9.1.1 Description**

Reads the clock value at the selected Station.

5.4.9.1.2 Call_Read_Clock

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 70							

Call_Read_Clock ::= RECORD

```

{}
-- no parameters

```

5.4.9.1.3 Reply_Read_Clock

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 70							
reserved1															
time_date (seconds)															
time_date (ticks)															

```

Reply_Read_Clock ::= RECORD
{
  reserved          WORD16 (=0)          -- for alignment
  time_date         TIMEDATE48           -- time value in seconds and
                                         ticks
}

```

5.4.9.2 Write_Clock

5.4.9.2.1 Description

Sets the clock at the selected Station.

5.4.9.2.2 Call_Write_Clock

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 71							
reserved1															
time_date (seconds)															
time_date (ticks)															

```

Call_Write_Clock ::= RECORD
{
  reserved1        WORD16 (=0)          -- for alignment
  time_date         TIMEDATE48           -- time value to set in seconds
                                         and ticks
}

```

5.4.9.2.3 Reply_Write_Clock

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 71							

```

Reply_Write_Clock ::= RECORD
{}
-- no parameters

```

5.4.10 Journal Service

5.4.10.1 Read_Journal

5.4.10.1.1 Description

This service reads the last “j” entries in the journal. The meaning of the entries is application-dependent. The handling of the index number is explained under the object description.

5.4.11 Equipment Service

5.4.11.1 Read_Equipment

5.4.11.1.1 Description

This service retrieves a pointer to a memory domain where a complete description of the supported equipment is located. The format of this data structure is outside the scope of the standard.

5.4.11.1.2 Call_Read_Equipment

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 82							
reserved1								reserved2							

Call_Read_Equipment ::= RECORD

```
{
  reserved1      WORD8 (=0),      -- reserved
  reserved2      WORD8 (=0),      -- reserved
}
```

5.4.11.1.3 Reply_Read_Equipment

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tnm_key								sif_code = 82							
reserved1								number_entries							
equipment_list ARRAY [number_entries] OF															
equipment_name: STRING32															
(CHARACTER8)								'00'H							
equipment_root															
equipment_size															

Reply_Read_Equipment ::= RECORD

```
{
  reserved1      WORD8 (=0),
  number_entries  UNSIGNED8,      -- number of returned entries
  equipment_list  ARRAY [number_entries] OF
  {
    equipment_name  STRING32,      -- identifies the equipment
    equipment_root  UNSIGNED32     -- base address of domain
    equipment_size  UNSIGNED32     -- size of equipment descriptor
  }
}
```

5.5 Interface Procedures

The interface procedures are split between a Manager Interface and an Agent Interface.

5.5.1 Manager interface (MGI)

All services of the Manager interface are provided by two generic procedures:

- a service request procedure `mm_service_req`, and
- a service confirm procedure `mm_service_conf`.

The Manager interface procedures are prefixed by `mm_`.

Description	Calls a remote service.	
Syntax	<pre>MM_RESULT mm_service_req (UNSIGNED8 station_id; const AM_ADDRESS* agent_adr; struct MM_CALL * mm_call);</pre>	
Input	<code>station_id</code>	Station_Identifier of this Station
	<code>agent_adr</code>	Network_Address of the Agent
	<code>mm_call</code>	mm_call identical to the format of the body of the Management Call Message. (The format of that structure depends on the SIF_code.)
Result	result of the call is an error code <code>MM_RESULT</code> , defined in 5.4.1.5.	

Description	The service confirm procedure <code>mm_service_conf</code> return the result of the service call to the Manager. This procedure may be a polled procedure or an indication procedure.	
Syntax	<pre>MM_RESULT mm_service_conf (UNSIGNED8 station_id; AM_ADDRESS * agent_adr; struct MM_REPLY * mm_reply);</pre>	
Output	<code>station_id</code>	Station_Identifier of this Station
	<code>agent_adr</code>	Network_Address of the Agent
	<code>mm_reply</code>	in case <code>MM_RESULT</code> is OK, this structure returns the body of the Reply_Message, otherwise, it is undefined. (The format of that structure depends on the SIF_code.)
Result	result of the call is an error code <code>MM_RESULT</code> , defined in 5.4.1.5.	

5.5.2 Agent interface

5.5.2.1 Description

The Agent Interface procedures however do not define the interface between Agent and Network, but between the Agent and the other processes of the Station.

The Agent interface gives a user access to the Agent, to poll two conditions:

- change (are modifications to the Station currently allowed ?)
- stop (is it allowed to stop the Station ?)

The Agent may need access to the real-time kernel to co-ordinate the user task execution.

Agent interface procedures are prefixed by ma_xxx.

5.5.2.2 Agent control procedures

5.5.2.2.1 Procedure ma_ask_permission

Description	Allows a user to poll which management requests exist. The application using this function will respond with ma_permit.		
Syntax	<pre> MA_PERMISSION ma_ask_permission (UNSIGNED8 task_id); </pre>		
Input	task_id	calling task (identified by array index of the task in tasks_list of Reply_Read_Tasks_Status)	
Output	-		
Return	MA_PERMISSION	0: MA_CHANGE_REQU, 1: MA_CHANGE_NOREQU, 2: MA_STOP_REQU, 3: MA_STOP_NOREQU	modifications requested no modifications requested stop request no stop request

5.5.2.2.2 Procedure ma_give_permission

Description	The application responds to the change request with this procedure, indicating whether change is allowed or not.		
Syntax	<pre> void ma_give_permission ENUM8 (decision); </pre>		
Input	decision	0: MA_CHANGE_ALLOWED, 1: MA_CHANGE_DENIED, 2: MA_STOP_ALLOWED, 3: MA_STOP_DENIED	change allowed no change permitted task may be stopped task may not be stopped
Output	-		

5.5.2.3 User service subscription

5.5.2.3.1 Type MA_SERVICE_CALL

Description	Type declaration of a procedure to be called for a given service call, which returns the parameters needed for the Reply_Message.	
Syntax	<pre>typedef void (* MA_SERVICE_CALL) (AM_ADDRESS * manager_address, void * call_msg_adr, UNSIGNED32 call_msg_size, void * * reply_msg_adr, UNSIGNED32 * reply_msg_size MM_RESULT * agent_status);</pre>	
Input	manager_address	pointer to the full network address of the calling manager.
	call_msg_adr	pointer to the start of the service Call_Message to be processed (tnm_key field).
	call_msg_size	size of the Call_Message in octets
	reply_msg_adr	pointer to the start of the service Reply_Message to be returned, (tnm_key field)
	reply_msg_size	size of the Call_Message in octets
	agent_status	result to be communicated as Agent status to the Manager.

5.5.2.3.2 Type MA_SERVICE_CLOSE

Description	Type declaration of a procedure to be called for closing a service.	
Syntax	<pre>typedef void (* MA_SERVICE_CLOSE) (void);</pre>	
Input	undefined	user-defined

5.5.2.3.3 Procedure `ma_subscribe`

Description	Indicates which user procedure to call when a user-defined service call is received. A previously assigned <code>SIF_code</code> is overwritten without a warning.	
Syntax	<pre>MM_RESULT ma_subscribe (ENUM16 command; ENUM8 sif_code; MA_SERVICE_CALL service_call; MA_SERVICE_CLOSE service_close void * service_desc);</pre>	
Input	<code>command</code>	0: subscribe 1: unsubscribe
	<code>sif_code</code>	user <code>SIF_code</code> (≥ 128)
	<code>service_call</code>	procedure variable of type <code>MA_SERVICE_CALL</code> which will execute the service when called.
	<code>service_close</code>	procedure which the Agent will call when the <code>Reply_Message</code> has been completely sent (e.g. to free the buffer).
	<code>service_desc</code>	descriptor of the service, as a visible string terminated by a '00'H character.
Return	<code>MM_RESULT</code>	

5.5.2.4 Restart procedure subscription

5.5.2.4.1 Type `MA_STATION_RESTART`

Description	Type declaration of a procedure to be called for restarting the Station after a time-out or a restart command. This procedure will probably not return.	
Syntax	<pre>typedef void (* MA_STATION_RESTART) ();</pre>	

5.5.2.4.2 Procedure `ma_subscribe_restart`

Description	Indicates which user procedure to call when the Station is reset or a reservation time-out takes place.	
Syntax	<pre>MM_RESULT ma_subscribe_restart (MA_STATION_RESTART station_restart);</pre>	
Input	<code>station_restart</code>	procedure which the Agent will call when the reservation time-out elapses or when a reset command has been received.
Return	<code>MM_RESULT</code>	

Annex A

(informative)

Tutorial on the TCN architecture

A.1 Introduction

This annex describes the Train_Communication_Network as a tutorial with informative status and may clarify the normative clauses 1 to 5. In case of divergency, the normative clauses take precedence.

A.1.1 Summary

The Train Communication Network (TCN) interconnects programmable equipment on-board rail vehicles for the support of:

- traction and vehicle control (remote control, doors, lights, ...);
- remote diagnostics and maintenance;
- passenger information and comfort.

The Train Communication Network encompasses two busses:

- the Multifunction Vehicle Bus (MVB) which interconnects devices within a vehicle, a bus optimised for fast response, operating at 1,5 Mbit/s on twisted wires or optical fibres;
- the Wire Train Bus (WTB) which interconnects the vehicles of a train, a bus capable of self-configuration, operating over twisted wires at a speed of 1 Mbit/s.

These busses offer the same two services at the Link Layer:

- Process Data: cyclic, source-addressed broadcast data;
- Message Data: on-demand, destination-addressed datagrams.

The higher-level, Real-Time Protocols offer two bus-independent application services:

- Variables (distributed process data base);
- Messages (Call/Reply and Multicast messages).

The Network Management supports configuration, maintenance and operation.

A set of Guidelines for Conformance Test supports interoperability of the devices.

Table A.1 summarises the characteristics of the Train_Communication_Network.

Table A.1 - Summary of the Train Communication Network

Characteristics	Wire Train Bus	Multifunction Vehicle Bus
Structure	variable structure, self-configured when composition changes	fixed structure and device addresses
Medium	shielded, twisted wire pair, 120 • (860 m, 32 Nodes, corresponding to 22 UIC vehicles)	twisted wire pair, biased RS-485; (20 m, 32 devices); shielded twisted wire pair isolated by transformer (200 m, 32 devices); optical glass fibre with star coupler (2000 m, 2 devices).
Physical redundancy	duplicated physical medium	duplicated physical medium
Signalling	Manchester with 16..32 bit preamble.	Manchester with delimiter
Signalling data rate	1,0 Mbit/s	1,5 Mbit/s
Address space	8-bit address for: • Process Data (one per Node); • Message Data	12-bit address for: • Process Data (Logical Address) • Message Data (Physical Address)
Physical address	point-to-point and broadcast	point-to-point and broadcast
Useful Frame size	variable: 4 - 132 octets	quantified: 16, 32, 64, 128 or 256 bits
Integrity	FCS-16 per frame, frame size check and Manchester encoding	IEC 60870 Check Sequence and frame size check
Medium allocation	by one Master	by one Master
Mastership transfer	each Node can become Master, Strong Master (by command) or Weak Master (by default)	Bus Administrators become Master upon token transfer
Master redundancy	mastership transfer to another Node after Inauguration	automatic mastership transfer redundancy check by token passing
Link_Layer services	Process Data cyclic Message Data sporadic Supervisory Data sporadic/cyclic	broadcast source-addressed Datasets point-to-point or broadcast datagrams data for bus management

Common Traffic	Variables	Messages
Link Layer Services	unconfirmed broadcast of source-addressed data (Process Data)	destination-oriented, connectionless datagrams (Message Data)
Link Control	overwritable Port with freshness supervision	non-overwritable Queues for packets
Network Layer	application task in the gateway copies data from bus to bus	origin and final address in each datagram, routing and directories
Transport Layer	-	connection-oriented, end-to-end flow control and error recovery.
Session Layer	-	Call/Reply_Message pairing
Presentation Layer	unified data types	unified data types
Application Interface	procedures for Process Variable access (individual, set or cluster)	procedures for Call/Reply and Multicast Messages

A.1.2 Users Requirements

The following user requirements were compiled during specification of the Train Communication Network:

A.1.2.1 Data classes

The Train Communication Network will transport two classes of data:

- time-critical, short Process_Variables (such as used for traction control);
- less urgent, but possibly lengthy Messages (such as used for diagnostics).

A.1.2.2 Timing

The maximum delay of a Process_Variable transmission through the whole network will be bound by a limited value:

- Process_Variables will be transmitted periodically;
- The TCN will allow the transmission of all highest-priority Process_Variables from application to application between Vehicle_Bus to Vehicle_Bus, transiting via the Train_Bus, in less than 100 ms;
- The TCN will allow to transmit all highest-priority Process_Variables from application to application between two devices on the same vehicle in less than 50 ms.

NOTE Actual delay may fluctuate, but under regular conditions will not exceed that value.

A.1.2.3 Topology

The Train Communication Network will base on a two-level hierarchy including:

- a Train_Bus interconnecting Nodes in the different vehicles;
- a Vehicle_Bus interconnecting the equipment within a vehicle.

A typical configuration is shown in Figure A.1.

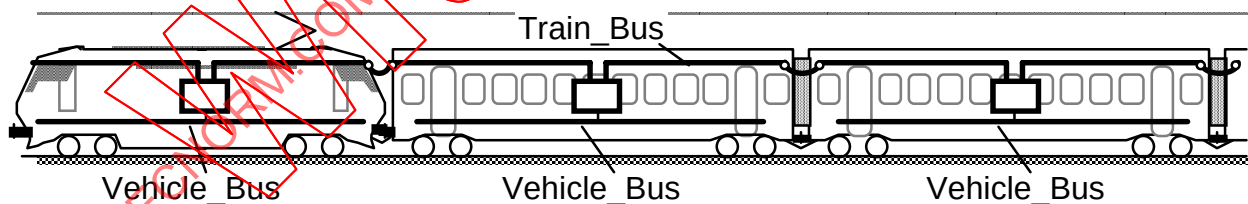


Figure A.1 - Train_Bus and Vehicle_Busses

The Train_Bus and the Vehicle_Bus will be connected over a Node which acts as gateway.

This hierarchy is conceptual: Nodes without a Vehicle_Bus, or Nodes to which several Vehicle_Busses are attached will be permitted.

NOTE The architecture could be extended by a sensor/actor bus as a third level of hierarchy. This level is not subject to standardisation in the Train_Communication_Network, since it is considered to be part of the Vehicle_Bus device to which the sensor bus is attached.

A.1.2.4 Train_Bus

The Train_Bus will be used to interconnect vehicles in trains of variable composition:

- The Train_Bus will support the train composition specified by the UIC leaflet 556, which consists of 22 vehicles, and in which the bus spans a distance of 860 m;
- The Train_Bus will support at least 32 Nodes;
- The Train_Bus Nodes will receive a position address and recognise their orientation (right/left, front/rear) within the train, as well as the position of the other Nodes;
- The Train_Bus will operate when two or more vehicles fitted with compatible equipment are electrically coupled, even if vehicles in between act only as continuity vehicles;
- The Train_Bus will tolerate coupling and uncoupling of vehicles at a rate of about one per hour;
- Since Nodes at the extremity or in the middle of the train could be inoperative due to damage or lack of power, the Train_Bus will inform the application on all vehicles about the number and type of Nodes to verify that it matches the train composition;
- The Train_Bus will be able to continue its operation without manual intervention when the number of vehicles changes or when devices leave or join the bus;
- The Train_Bus will use a dedicated medium;
- This medium will be specified in such a way as to permit its inclusion into the UIC cable (standardised cable for loudspeaker, lights and doors control as described in UIC leaflet 568) or the EP cable (Electro-Pneumatic brake cable as described in UIC leaflet 648).

NOTE Originally, it was foreseen to use certain wires of the UIC cable as a medium. The achievable performance was however too low.

A.1.2.5 Vehicle_Bus

The Vehicle_Bus will be used to interconnect equipment within a vehicle or an inseparable group of vehicles:

- The Vehicle_Bus will support devices located up to 200 m apart;
- The Vehicle_Bus will support at least 256 devices;
- The Vehicle_Bus will provide a response time below 16 ms in the worst case.

A.1.2.6 Availability

The single failure of a piece of equipment will not prevent unrelated equipment from working. This applies especially to:

- occasional corruption or loss of data;
- medium interruption or short-circuit;
- continuous transmission by a damaged transmitter;
- denied medium access;

- configuration error;
- loss and reintegration of Nodes.

The disruption of the Train_Bus, in case of composition change or device failure/reintegration, will remain below 1,0 second for 22 vehicles (applications have been identified which require less than 0,3 second for 4 vehicles).

A configuration able to provide communication in spite of the failure of any single piece of equipment will be possible.

The level of redundancy will depend on the actual availability requirements of the application.

A.1.2.7 Environmental Condition

The Train Communication Network devices will comply with the standards for railways electronic equipment, in particular with IEC 60571.

A.1.2.8 Classes of Busses

The Train_Bus and the Vehicle_Bus provide the same data transmission services. They have technical differences which originate from their different roles:

- a Train_Bus for Open_Trains can name its Nodes and determine their orientation. This requires additional hardware and software, superfluous in Vehicle_Busses;
- the Vehicle_Bus has stricter timing requirements than the Train_Bus. The response time is about ten times faster on the Vehicle_Bus than it is on the Train_Bus.

The basic differences between Vehicle_Bus and Train_Bus are summarised in Table A.1 (at the beginning of this document).

A.2 Tutorial on the Real-Time Protocols

This annex describes informally the Real-Time Protocols (RTP). The corresponding specification appears in Clause 2.

A.2.1 Train Communication Network services

The Real-Time Protocols (RTP) are the protocols and services offered to an application for communicating with another application over the Train_Communication_Network. The application could be in another vehicle, within the same vehicle or within the same device.

The Real-Time protocols may be used for communicating over the Multifunction Vehicle Bus, the Wire Train Bus or any other bus which offers the same basic services.

Figure A.2 shows a train consisting of vehicles of different types, one with MVB, one with an unspecified vehicle bus and one without a vehicle bus. The train bus could be the WTB or another bus. If all devices obey the Real-Time Protocols, no protocol conversion is needed.

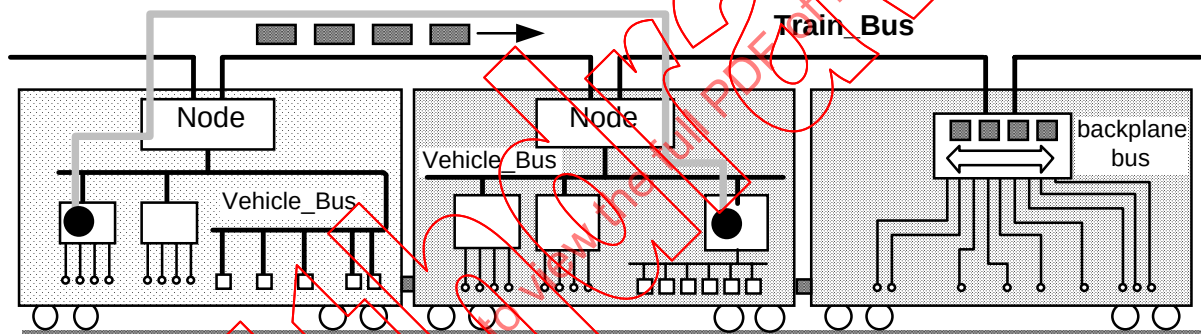


Figure A.2 - Data transfer over the Train Communication Network

A.2.1.1 Periodic traffic and sporadic traffic

The busses in the TCN support two types of data transmission: Periodic_Data and Sporadic_Data:

- Periodic_Data are transmitted cyclically, at a period which is a multiple of a Basic_Period, during the Periodic_Phase. The Periodic_Phase occupies a fixed portion of every Basic_Period. During that time, the Master polls the devices for Periodic_Data in a predefined sequence. This traffic is called the periodic traffic;
- Sporadic_Data, if any, are transmitted on demand, during the interval between two Periodic_Phases, called the Sporadic_Phase. This traffic is called the sporadic traffic.

An example of a time slice on a bus is shown in Figure A.3.

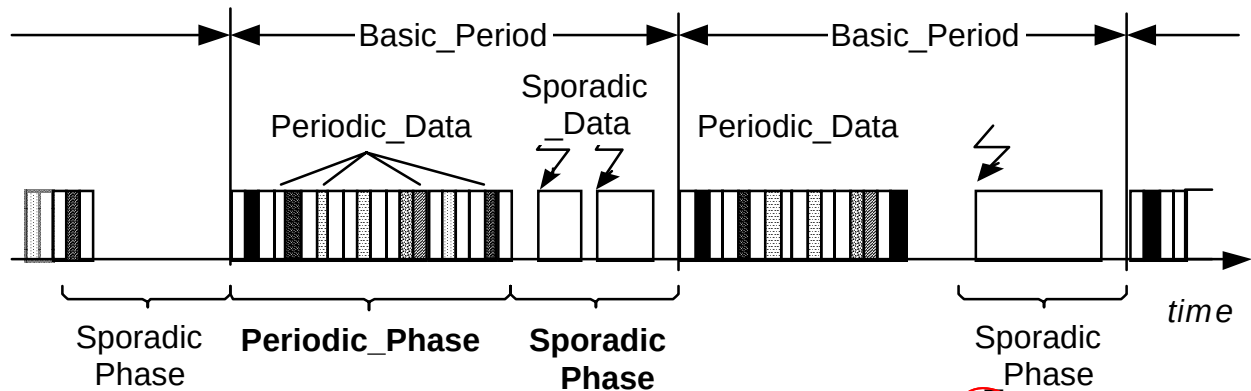


Figure A.3 - Example of Periodic_Data and Sporadic_Data transmission

On the MVB and WTB, the bus always remains under control of the Master. Slaves cannot send spontaneously, even during the Sporadic_Phase. This is to maintain the deterministic character of Process_Data transmission.

A.2.1.2 Transmitted Data

The Train_Bus or the Vehicle_Bus transports three types of data:

1) Process_Data:

- Process_Data are data broadcast by a Publisher device to a number of Subscriber devices, and which are identified by their Logical_Address;
- This kind of transmission is called source-addressed broadcast.;
- To each Process_Data corresponds a source buffer (Port) in its Publisher device and a sink buffer (Port) in every Subscriber device;
- The Master requests transmission of given Process_Data from the source buffer by calling their Logical_Address;
- The devices subscribed to this Logical_Address insert the received data into their buffer, overwriting the previous value with the transmitted value;
- Process_Data are transmitted periodically, as Periodic_Data;
- Process_Data carry the Datasets used to provide the Process_Variable service.

2) Message_Data:

- Message_Data are destination-oriented data sent from a source device to a destination device or all devices within the same bus. Each device holds one pair of queues for all Message_Data it sends or receives. The received Message_Data are inserted into the input queue of the destination device if the queue has room for it;
- Message_Data are transmitted on demand;
- Message_Data carry the Packets used to provide the Message service.

3) Supervisory_Data:

- Supervisory_Data are data used to supervise the status of the devices, to detect silent devices, to transfer mastership, to inaugurate the bus, etc.. within the same bus;
- Some Supervisory_Data are transmitted periodically, other on demand.

NOTE 1 Periodically transmitted Process_Data are called Regular Data in the UIC 556 leaflet.

NOTE 2 Process_Data can also be transmitted on-demand, as Sporadic_Data, but this is not supported by this Standard since such a service is not reliable.

NOTE 3 Sporadically transmitted Messages are called Event Data in the UIC 556 leaflet.

NOTE 4 Message_Data can also be transmitted periodically, but this is not supported by this Standard.

A.2.1.3 Variables and Messages

The three Link_Layer services (Process_Data, Message_Data and Supervisory_Data), with their associated transmission mode (periodic and sporadic), provide three basic services: Variables, Messages and Network_Management as shown in Figure A.4.

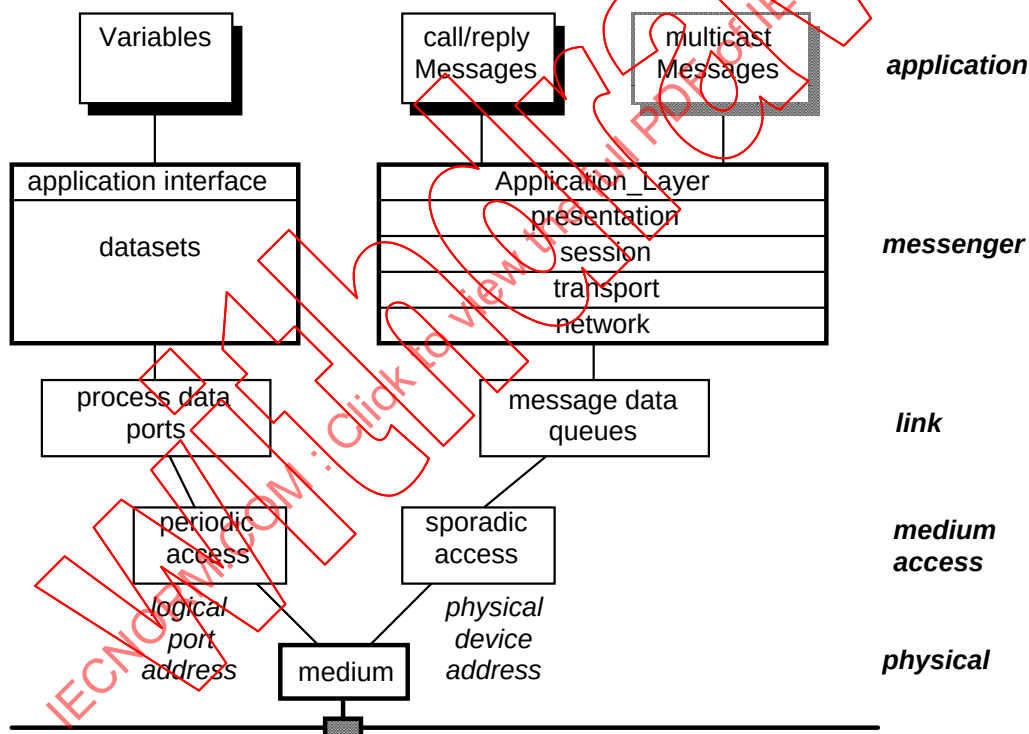


Figure A.4 - TCN Services

The Application_Processes (implemented, for instance, by tasks) running on a Node have often the same operating mode as the TCN services they rely on. For instance, Process_Variables are preferably consumed and produced by periodic processes, as is common for control algorithms. Messages, conversely, are exchanged between event-driven processes, as is common for diagnostic tasks as shown in Figure A.5.

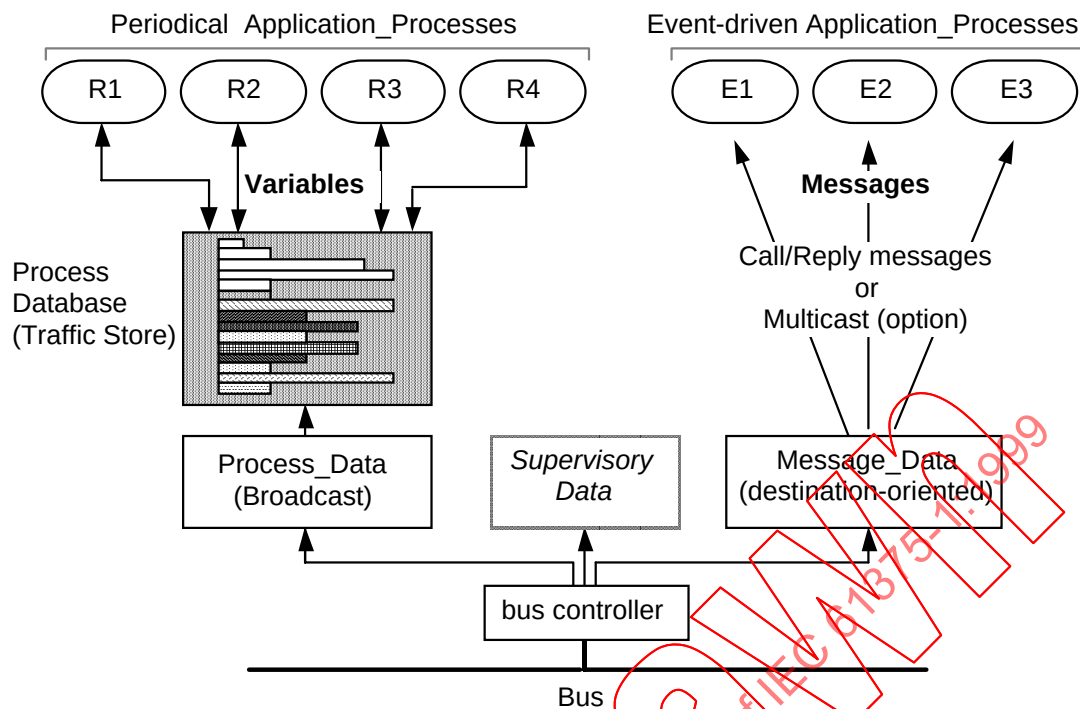


Figure A.5 - Application Tasks and TCN services

The Real-Time Protocols are layered similarly to the OSI-layering, as shown in Figure A.6.

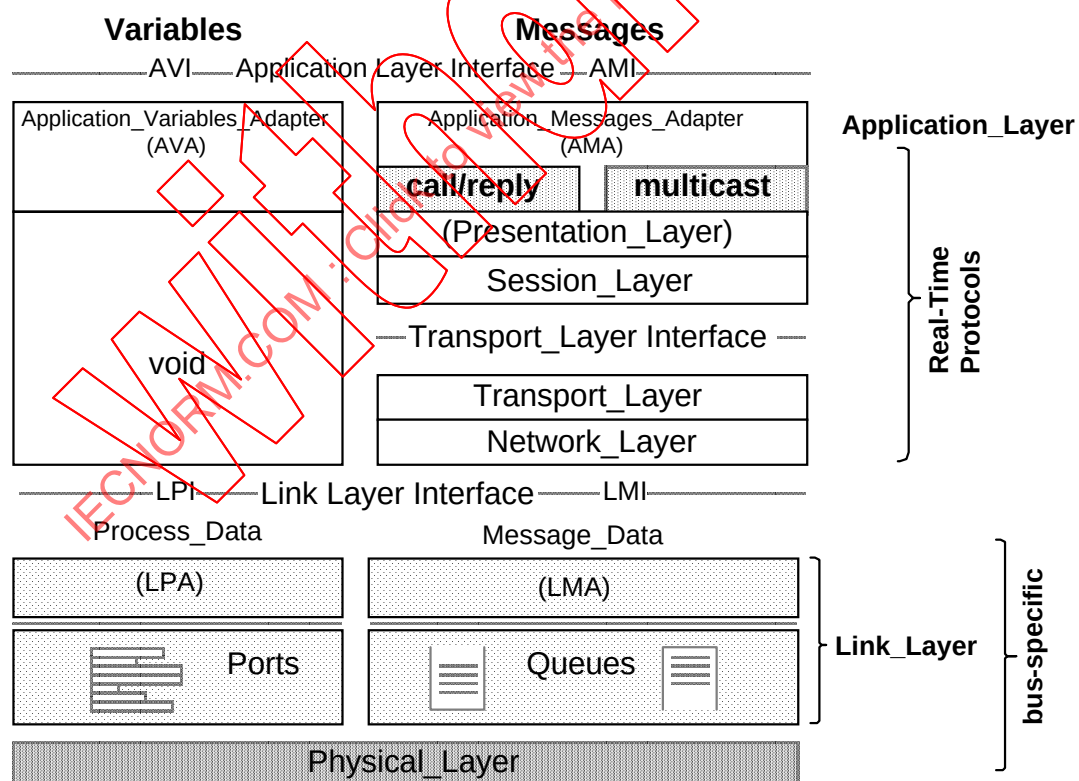


Figure A.6 - Layering of the Real-Time Protocols

This Standard specifies two interfaces:

- the Link_Layer Interface (LPI and LMI), which define which services are expected from the bus (whichever is used);
- the Application_Layer Interface (AVA and AMI), which define, respectively, the Variables and the Messages Services offered to the Application.

This Standard specifies also the corresponding protocols and the encoding of the transmitted data.

NOTE The Transport_Layer Interface for messages is defined for future use.

Table A.2 summarises the Real-Time Protocols.

Table A.2 - Summary of the Real-Time Protocols

	Variables	Messages
Application	individual access cluster access set access	Call_Messages / Reply_Messages Multicast_Message
Presentation	unified primitive and structured variable types	
Session	not existent	conversation establishment pairing of Call_Message and Reply_Messages
Transport	not existent	message segmenting into packets, flow control and error recovery through sliding window protocol. in multicast messages, negative acknowledgement and retry.
Network	not existent	hierarchical addressing scheme, directories and routing
Services expected from the bus		
Link_Layer Control	Dataset, freshness supervision	connectionless Link_Layer
Medium Access	periodic transmission	sporadic transmission

A.2.2 Tutorial on the Variables Services

A.2.2.1 General

The Real-Time Protocols offer the "Variables" service for the transmission of time-critical data, or Process_Variables, over the Train Communication Network. The Variables Service relies on the ability of the busses to broadcast source-addressed data to all devices.

A.2.2.2 Process_Variables

A.2.2.2.1 Application view

Process_Variables represent the state of a physical process. They are transmitted for the purpose of monitoring, control or command.

For instance, Process_Variables carry the axle speed, motor current or brake force. They are displayed in the driver's cab, carry the driver's commands, control regulation loops, etc. Process_Variables are used in time-critical applications and should be delivered promptly.

The TCN offers a transmission service for Process_Variables using the concept of a Distributed Process Database.

In this concept, applications in the different devices access the Process_Variables as if they were located in a global database. Each device holds the Process_Variables it produces or consumes in a shared memory called a Traffic_Store, which is a local, partial copy of the global database produced by the network.

Thus a variable put into one of the Distributed Database is distributed to all other copies. The application which generates that variable is called the Publisher, the applications which receive that variable are the Subscribers. The bus is responsible for the actualisation of the databases.

Figure A.7 shows an example. When Application 3 (Publisher) modifies a Process_Variable, it places its value into its Traffic_Store and can then proceed with other tasks. After a while, the bus updates the databases.

To this effect, the source device's Bus_Controller reads the value and broadcasts it on the bus; the Bus_Controllers of the subscribed sink devices store the received value into their Traffic_Stores, overwriting the previous value. Some time later, Application 1 or Application 4 (Subscribers) fetches the value from the Traffic_Store.

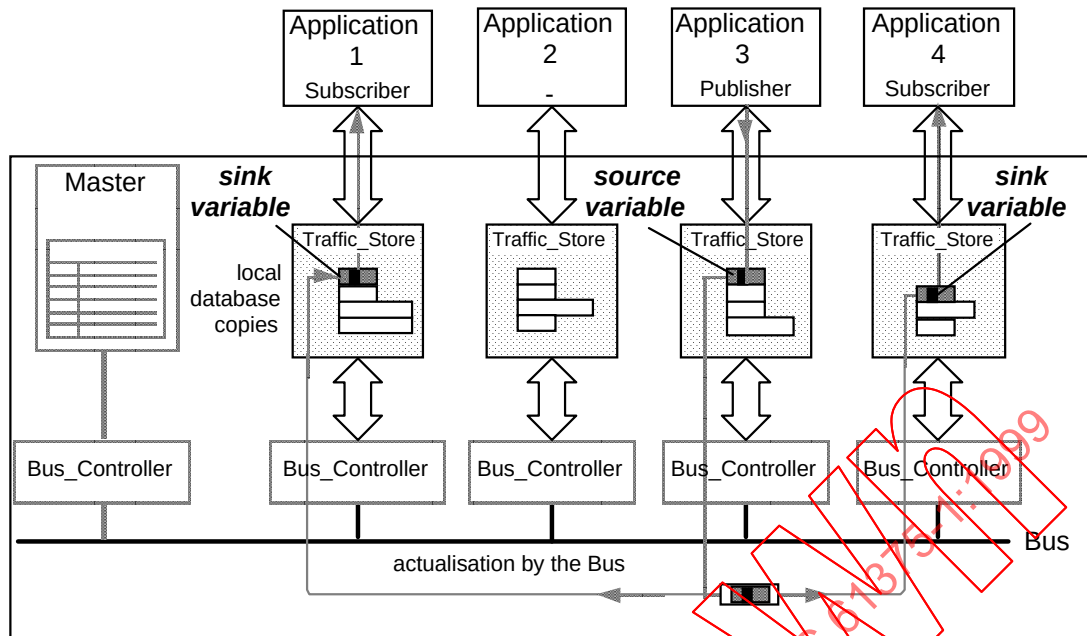


Figure A.7 - Variable transmission and Ports

All three entities: Publisher, Bus and Subscriber, operate asynchronously (they can be synchronised if required), and, in most cases, cyclically (but possibly with different periods).

A.2.2.2.2 Source-addressed broadcast

Process_Variables have a small size: binary variables are represented by one or two bits, most analogue variables by 16 bits. Since it is not efficient to transmit them individually, several Process_Variables generated by the same device are grouped for transmission into one Process_Data frame.

The broadcast ability of a bus provides an efficient actualisation of the Distributed Data Base. Since the same Process_Variable is often needed in several places, Process_Data are broadcast to enhance throughput. Process_Data are identified by their source, rather than by their destination address. This is called *source-addressed broadcast*.

Both the MVB and the WTB are controlled by one master, which controls the sending of the Process_Data.

The Master broadcasts a Master_Frame, which carries the identifier of a Process_Data frame. The device which sources these Process_Data responds by broadcasting a Slave_Frame, which is received by all other devices.

All devices interested in these Process_Data receive them from the bus as shown in Figure A.8.

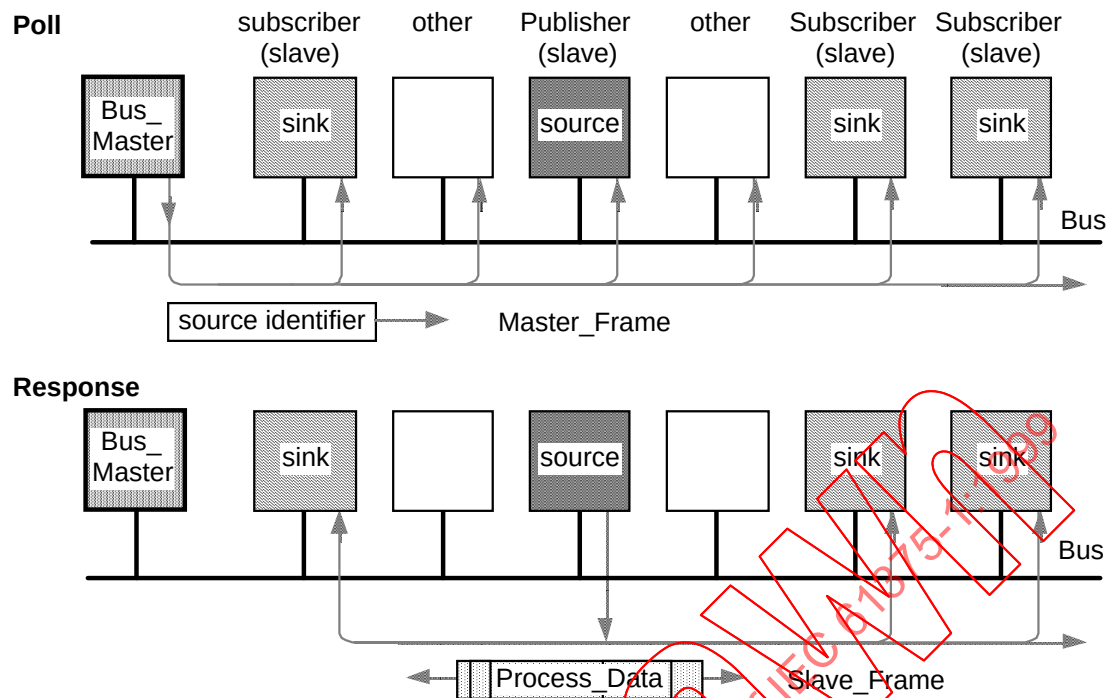


Figure A.8 - Broadcasting of source-addressed Process_Data

Since Process_Data are identified by their source, a configuration carried out before regular operation ensures that Master, source and sink agree on the format and meaning of the data associated with a given identifier.

The Master does not usually participate in the data exchange, it just plays the role of an intermediary. The device acting as a Master may also operate as a Slave and participate in the traffic as any other device.

A.2.2.2.3 Ports

The source Slave responds to the poll of the Master immediately, with no other device allowed to transmit in between. Therefore, the Process_Data response from the Slave comes very shortly after the master polled it (a few microseconds on the MVB). Similarly, the application may not be delayed for writing or reading the variables for more than a few microseconds.

Such a short reply delay requires a shared memory between application and network, and ensures that simultaneous access preserves consistency.

A source application places its Process_Data into a Port, which is a register holding a complete bus frame. The Master_Frame on the bus addresses one source Port and a number of sink Ports. The source Port and the sink Ports have the same identifier, although they belong to different devices.

A Port ensures that the data is read consistently when it is accessed simultaneously by the bus and by the Application_Processor. This requires that a Port consists of at least two pages, which are used alternatively.

A Port also provides means to control access, data freshness and interrupts. This may be implemented by a Port Control and Status register associated to each Port.

The Ports of a device are located in the *Traffic_Store*, a shared memory which can be accessed both by the *Application_Processor* (CPU) and by the *Bus_Controller*. In simple devices, the Ports are reduced to a register, and there is no *Traffic_Store*.

A typical *Traffic_Store* with eight Ports is shown in Figure A.9.

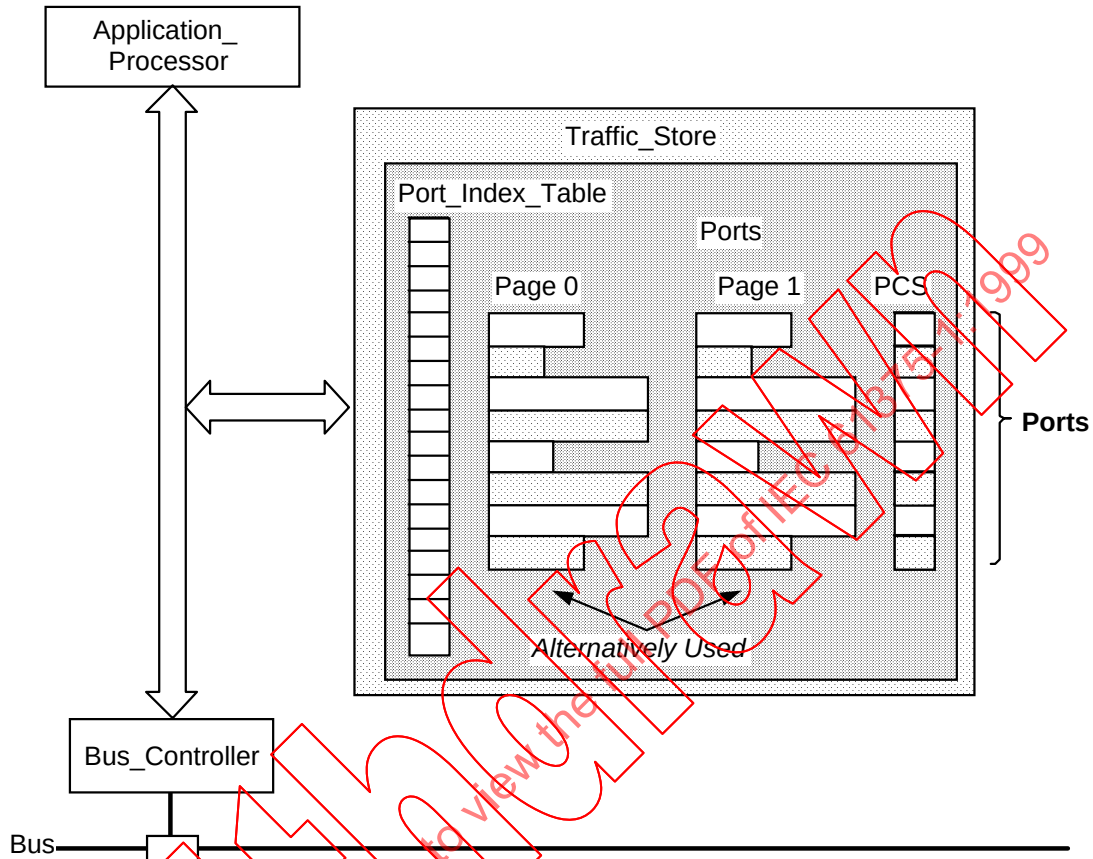


Figure A.9 - Port and Traffic_Store

The *Traffic_Store* includes means to address the Port. Since a device can be a subscriber to a large number of sink and source Ports, the decoding of the Port address received on the bus is performed by a look-up table, called the *Port_Index_Table*.

A.2.2.2.4 Process_Data medium access

Since *Process_Variables* are used in time-critical control loops, their transmission should respect strict timing. To this effect, *Process_Data* are transmitted periodically, with the frequency depending on their urgency.

Process_Variables may be transmitted periodically or on demand. Periodic transmission transmits the value, whether the value has changed or not. By contrast, aperiodic transmission only transmits data when its value changes.

- Periodic transmission may seem a waste: *Process_Variables* are sampled even when their value has not changed since the last sampling. For instance, a fire sensor, which may never be activated, is nevertheless sampled many times a second.

Alarms cannot interrupt the processing for urgent treatment, they should wait their turn. Periodic transmission, however, allows the transport of variables reliably under worst case conditions, when all variables change at the same time.

The response time is bounded, exact sampling is required. Obsolete data can be dangerous.

The disadvantage of Periodic_Data is that they occupy a fixed portion of the bus bandwidth. They are relatively inflexible since the frame structure may not vary during operation. This imposes some restrictions on the data formats.

- Aperiodic_Data are transmitted on demand. This transmission is related to *events*. While it is relatively easy to declare, as an event, the change of value of a binary variable, this is more difficult with analogue variables.

The building of events require application-specific knowledge. Therefore, Aperiodic_Data are transmitted in messages under the control of the application.

The advantage of Aperiodic_Data is that they only use bus bandwidth when they need transmission. Extension for future applications is relatively easy.

By contrast, their delivery delay cannot be guaranteed under worst-case conditions, since they are transmitted on demand.

The difference between Periodic_Data and Aperiodic_Data is most visible in the recovery procedure:

- If Periodic_Data are lost, no recovery is required. Occasional loss of data is not a problem: the next period will bring a fresh value, and lost data look like noise. This implies that only states of variables should be transmitted, not changes of state;
- If Aperiodic_Data are lost, the missing item should be repeated. This requires an acknowledgement for each transmission.

In both Periodic_Data or Aperiodic_Data, the receiver should check that Process_Variables are periodically refreshed, so the application can take action on obsolete data. This implies some refreshing even for Aperiodic_Data. It may for instance, be necessary to rebuild an image of the whole state of a vehicle at initialisation time or after a disruption.

For these reasons, the Train Communication Network supports periodic transmission of Process_Variables with a fixed period. When Aperiodic_Data are needed, the Train Communication Network offers the Messages service under the user's responsibility.

A.2.2.3 Process_Data protocol

A.2.2.3.1 Flow control

No Flow control is needed since the subscriber retains only the most recent value. The previous contents of the Port are lost if it has not been read in the meantime by the application. The application is not obliged to fetch every value.

A.2.2.3.2 Error recovery

Error recovery is implicit since there is no need for a retransmission; a lost sample will be refreshed by the next transmission. The application should tolerate infrequent loss of samples.

A.2.2.3.3 Freshness supervision

When a device fails, it ceases to produce new values. The sinks operate with the former value which becomes obsolete. To uncover this situation, the Link_Layer provides a freshness supervision associated with each Dataset.

When the application reads a Dataset, the Port indicates how long ago it has been updated. The application decides on the action to undertake if the variable is stale. For instance, if an input value has not been updated for a time greater than 4 seconds, the application may switch the output to a powerless or safe value.

Supervision is implemented in simple devices by a hardware timer. In devices with a large number of Ports, a background task updates the freshness counters of the Port.

A.2.2.3.4 Synchronisation

The Bus, the Publisher Application Process and the Subscribers Application Processes operate independently. There are some applications which need to be informed that a particular Dataset has been sent or received, for instance to synchronise the Vehicle_Bus and the Train_Bus operation in a gateway.

To this effect, it is possible to attach an indication procedure to a Port.

This, however, is not intended to introduce a sporadic transmission of Process_Data, since Ports are still polled cyclically.

A.2.2.3.5 Differences between busses

- in the Multifunction_Vehicle_Bus, one Process_Data frame contains the data of exactly one Port. A device holds typically 256 Ports. The total number of source Ports on a bus is limited to 4095. Special devices such as monitors may hold 4095 sink Ports. The (logical) address of a Port is not related to the device's address (except for field devices);
- in the Wire_Train_Bus, the Port address is tied to the Node_Address. A Node has one source Port, which holds all Process_Variables published by that Node, and one sink Port for every other Node of the bus. A Process_Data frame, and therefore a Port, may hold up to 128 octets, as shown in Figure A.10.

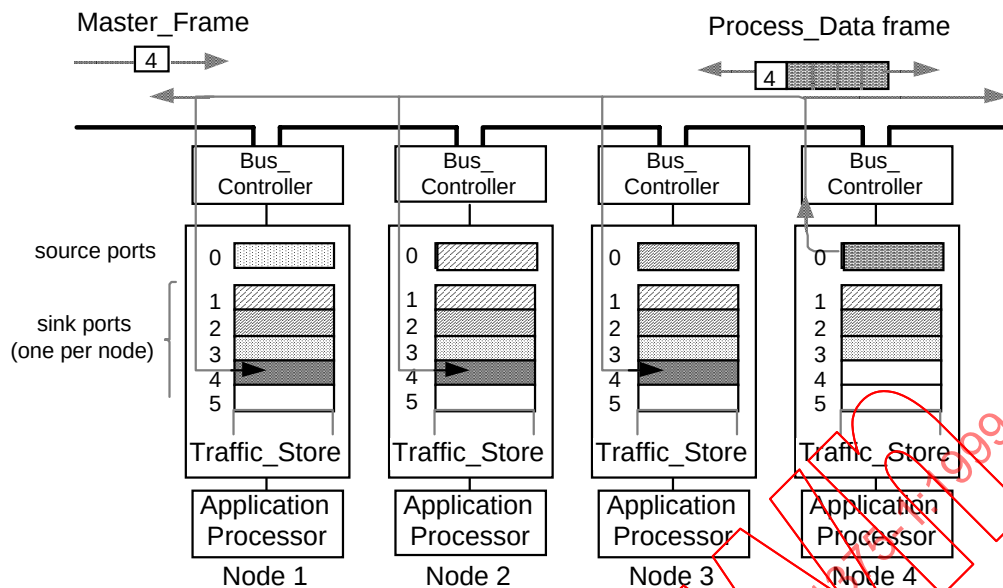


Figure A.10 - Ports on the Wire Train Bus

A.2.2.4 Datasets

Since most Process_Variables are small items, it is economical to pack several of them for transmission in the same frame. Packing is usual for binary data, which are preferably transmitted as a set of, for instance, 16 data bits. Packing is a necessity on the Train_Bus, since each Node sources only one Process_Data frame which broadcasts all Periodic_Data published by the Node. Process_Variables transmitted jointly form a Dataset.

The different Process_Variables of a Dataset may be consumed by different devices, but a Dataset is published by only one device.

A Dataset may have a size of between 16 and 1024 bits.

A Dataset contains only values but no addresses. Within a Dataset, a Process_Variable is identified by its Var_Offset, i.e. its bit position with respect to the beginning of the Dataset. Figure A.11 shows an example of a Dataset which mixes analogue and binary Process_Variables.

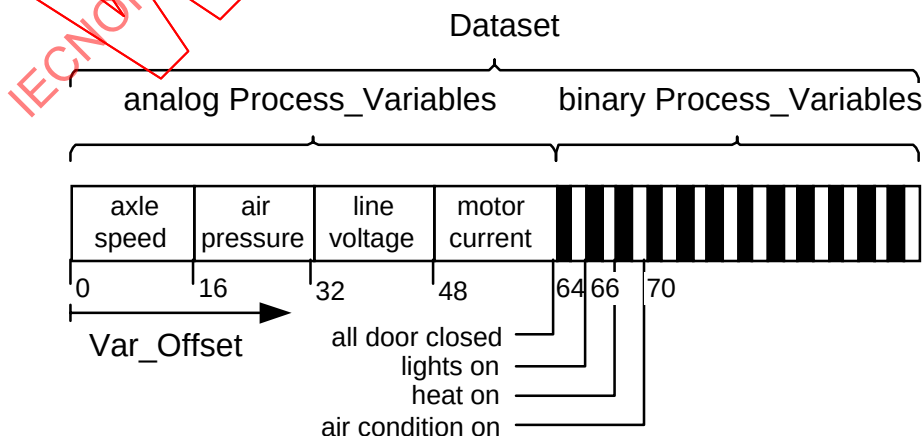


Figure A.11 - Dataset

In Figure A.11, the Process_Variable "lights on" is transmitted within the Dataset at Var_Offset = 66.

A Dataset may contain Process_Variables of different types, in any order as shown in Figure A.11.

A Dataset is consistent, i.e. all data or none are transmitted.

A Dataset is static: its format and size may not be modified without previously informing all Publishers and Subscribers. On the WTB, the Dataset format may change as a consequence of an Inauguration. On the MVB, the Dataset format is fixed at configuration time.

A Dataset may have reserved positions for future use. Filling these positions does not change the deterministic character of transmission, since both the Individual_Period and the frame size remain unchanged.

A Dataset may contain invalid data, for instance because the sensor delivering this Process_Variable failed, but the other Process_Variables in the Dataset are still correct.

In the case where it is important to check the validity of Process_Variables, a Process_Variable receives a validity indication, which specifies if it is erroneous or undefined.

This validity is indicated by a Check_Variable in the same Dataset, which is another Process_Variable, consisting of two bits normally antivalent, both bits should be different to indicate a valid value.

The Process_Variable and its Check_Variable in an MVB frame are shown in Figure A.12.

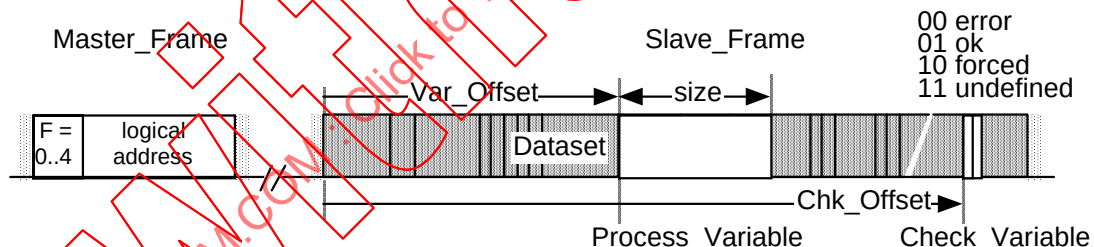


Figure A.12 - Validity bits

The Check_Variable protects against failures: in the worst case, if the network is unable to transmit a valid Dataset (for instance because its Producer application is not running anymore) a watchdog may overwrite the whole Dataset with "0", thereby setting all Check_Variables to "error".

The Check_Variable also ensures compatibility with future extensions: an older generation device overwrites the fields which are not yet used with "1". A device of a more recent generation, when reading this field, will interpret the variables as "undefined", but will not consider this as an error.

A.2.2.5 PV_Name

A Process_Variable is uniquely identified by its PV_Name, which includes the identifier of the Traffic_Store, the identifier of the Dataset within that Traffic_Store and the Var_Offset within the Dataset.

In addition, and as an optimisation for faster access, the PV_Name includes the type and the size of the Process_Variable, and the Chk_Offset of its Check_Variable.

The PV_Name of different instances of the same Process_Variable can differ only in their Traffic_Store Identifier.

Example - Figure A.13 shows an example with different instances of a Process_Variable.

On device X, the instance is identified for all Applications on that device by PV_Name = {Traffic_Store = 3, Port_Address = 1234 and Var_Offset = 8}.

On device Y, the instance is identified by PV_Name = {Traffic_Store = 1, Logical_Address = 1234 and Var_Offset = 8}.

For a bus analyser, the instance is identified as PV_Name = {Bus = 1, Logical_Address = 1234, Var_Offset = 8}.

All instances of the Process_Variable have the same Var_Offset and the same Logical_Address.

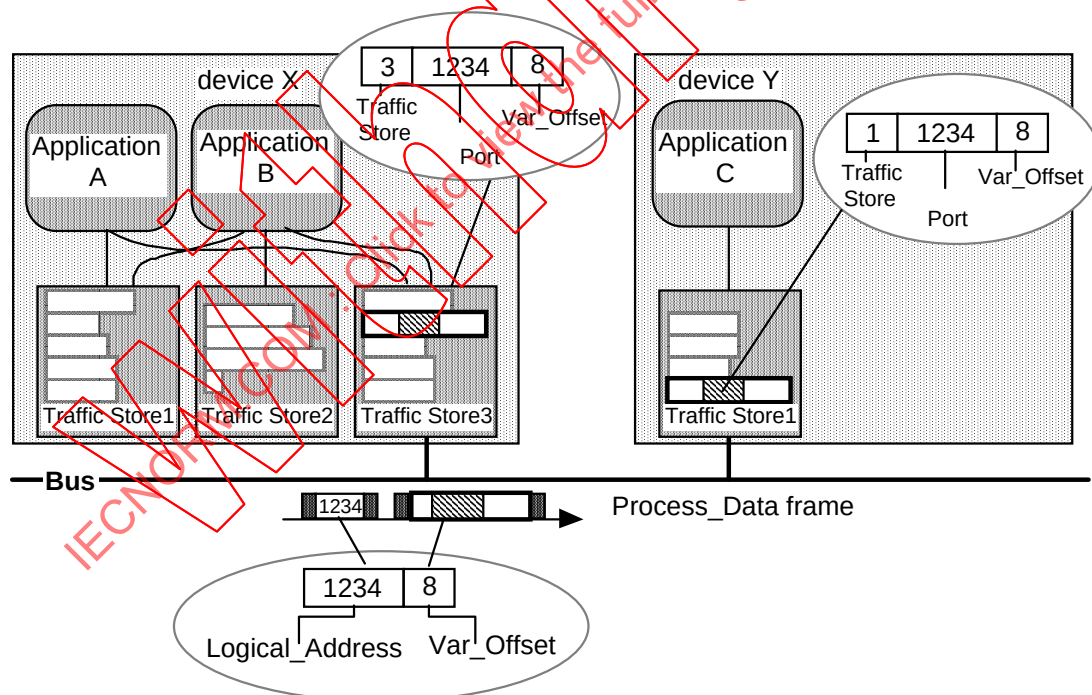


Figure A.13 - Multiple Process_Variable instances

In Figure A.13, Applications A and B on device X communicate over Traffic_Store 1 or 3.

A.2.2.5.1 WTB Dataset

On the Wire_Train_Bus, the maximum size of a Process_Data frame (or of a Port) is 1024 bits. A WTB Dataset has therefore a maximum size of 1024 bits. The application identifies a variable within a Dataset by its 6-bit Node_Address followed by its 10-bit offset.

Process_Variables on the Train_Bus need to be standardised for each specific application by the users, manufacturers or International Organisations such as UIC.

Different users will assign, to each category of equipment, a set of standard frames.

During Inauguration, each vehicle announces the equipment it supports. For each item of equipment, the frame type and their contents are standardised.

For instance, a vehicle with an attended driver's cab will broadcast a certain type of frame; a vehicle supporting leading motor equipment will broadcast frames of another format. If a vehicle, such as a locomotive, carries different types of equipment, it concatenates both the Process_Variables of each equipment in the frame.

As an example, the following frame sizes reflect a typical application:

- Attended driver's cab: 8 octets;
- Traction Leader: 14 octets;
- Traction Trailer: 9 octets;
- Coach: 9 octets.

If a vehicle combines Attended Driver's Cab, Traction Leader and Passenger Coach, a frame of 31 octets will be issued by that vehicle.

There are practical limitations to the frame size. At 1,0 Mbit/s, the transmission of the largest Process_Data set of 128 octets takes 1,4 ms. Polling 20 vehicles takes 28 ms, which is longer than the Basic_Period of 25 ms. Therefore, it is not possible to poll all vehicles every Basic_Period with the longest Dataset.

Several measures can reduce the load, such as:

- shorter frames;
- different priorities (Individual_Periods).

A.2.2.5.2 MVB Dataset

On the Multifunction_Vehicle_Bus, a Dataset has a size quantified to 16, 32, 64, 128 or 256 bits. The application identifies a Dataset uniquely by its 12-bit Port address and the Process_Variable within the Dataset by its 8-bit offset.

Given the high diversity of vehicles, the Dataset formats on the Vehicle_Bus are valid for one vehicle type only and cannot be standardised for all vehicles. The Dataset format is fixed by the hardware for field devices and configured by Network_Management for complex devices. The configuration is loaded at configuration time and does not change during operation.

A.2.2.5.3 Process_Variable instances

Process_Variables represent the actual state of a physical process.

The value of a Process_Variable is broadcast by its Publisher and distributed to one or several Subscribers, located on the same or on different devices.

Applications are the end users (Publisher or Subscriber) of Process_Variables. Conceptually, all Applications run in parallel, as separate processes (processes may be implemented as tasks, processes, threads or similar constructs).

Conceptually, all Applications access a Process_Variable as if it were located in a common memory, or a distributed database.

This common memory forms a distributed database, implemented in each device by a Traffic_Store, the data structure which stores the Process_Variables to be sent or received.

Several Process_Variables may be carried by the same Process_Data frame.

Although a Process_Variable exists conceptually only once, it has several instances: in the Publisher, in the Traffic_Store of the Publisher, on the bus, in the Traffic_Store of the Subscribers and in the Subscribers themselves as shown in Figure A.14.

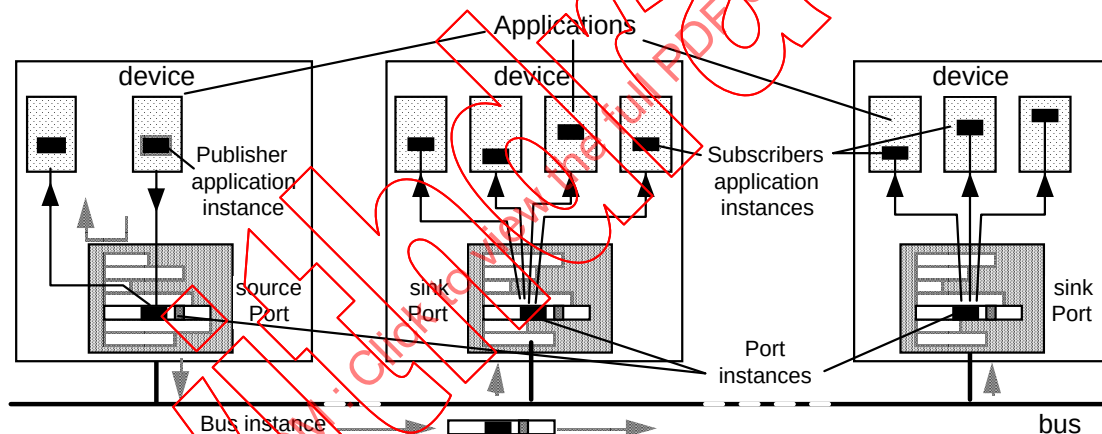


Figure A.14 - Instances of a Process_Variable

At any given time, the value produced at the Publisher may differ from the value transmitted over the bus and from the value present in the Traffic_Store. This is because Publisher and Subscribers are not synchronised among themselves and with the bus. The values read by the different Subscribers may therefore differ depending on the time of sampling.

A.2.2.6 Networking of Process_Variables

A.2.2.6.1 Gateway operation

Process_Variables have no Network_Address. Therefore the Network_Layer for Process_Variables is void. Process_Variables are transferred over bus boundaries by an application task. The vehicle and the Train_Bus are thus connected by a gateway, which interfaces to both the Vehicle and the Train_Bus as shown Figure A.15.

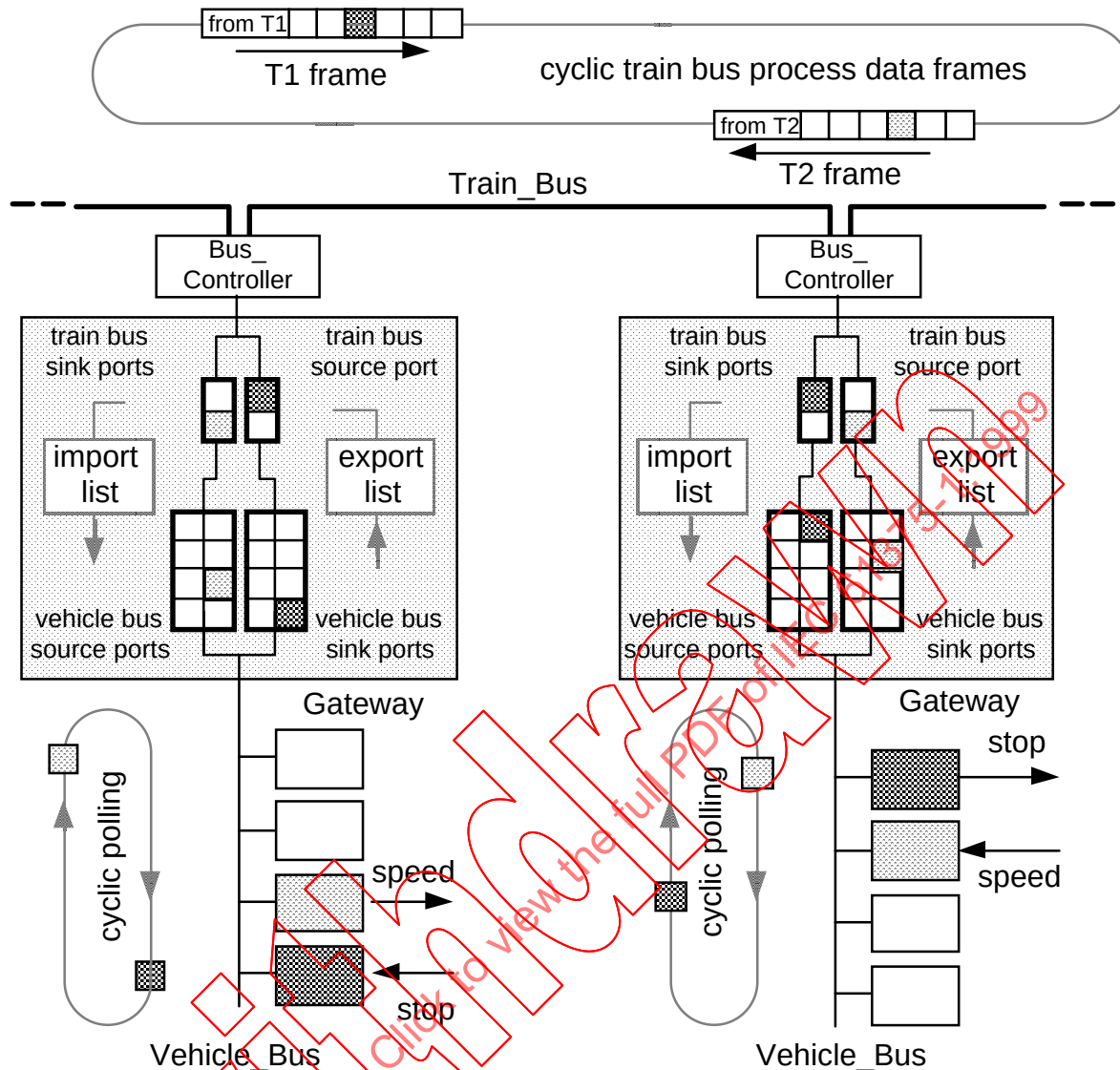


Figure A.15 - Transmission of Periodic Data through the network

Gateways are application-dependent, i.e. their configuration may be different for each application. This configuration tells the gateway which **Process_Data** should be transmitted from bus to bus.

In Figure A.15, a **Process_Variable** (e.g. the speed) is read in one **Vehicle_Bus** and transmitted to another **Vehicle_Bus** over the **Train_Bus**. Neither the vehicle nor the **Train_Bus** **Periodic_Data** frames carry a **Network_Address**. Therefore, the gateway observes the traffic on both the vehicle and the **Train_Bus** to identify the variables.

Not all **Vehicle_Bus** variables are exported - it would make little sense to let the **Train_Bus** carry the sum of the traffic of all **Vehicle_Busses**. The gateway should filter the **Process_Variables** to be transmitted.

The gateway acts as a **Vehicle_Bus** device subscribed to a set of export variables, which it receives over its **Vehicle_Bus** Port. It copies these **Process_Data** into the **Train_Bus** Port for publication. The gateway builds a frame containing the export variables, in the form of a broadcast frame containing its source (Node) address.

After a while, the gateway gains access to the Train_Bus and broadcasts the frame, which will be received at the gateway(s) of the other vehicle(s). The variables will be received there in the Train_Bus Port. The sink gateway filters out those import variables it is interested in and places them into the Vehicle_Bus Port for transmission at the next opportunity.

The transmission of data in the reverse direction (e.g. the stop signal) takes place in a similar manner.

This mode needs no event-driven operation on the Train_Bus, all busses operate cyclically.

Therefore, the intelligence of Process_Variable networking resides in the gateway marshalling task, which is configured by the application programmer with two lists of variables, the export and import lists ("import" is the point of view of the Vehicle_Bus). These lists can be loaded by Management_Messages.

The gateway application task may also process the data, for instance check that all doors in the train are closed, so as to forward only the result on the Vehicle_Bus.

A.2.2.6.2 Devices with several Traffic_Stores

A device may have up to 16 Traffic_Stores. These Traffic_Stores need not have the same structure, although this is advantageous.

A gateway accesses the Vehicle_Bus and the Train_Bus over their respective Traffic_Stores. However, any device, not only gateways, may have several Traffic_Stores. This permits Application_Processes residing within the same device to communicate in the same way as if they were located in different devices. An Application_Process may access several Vehicle_Busses, non-standard busses, such as a parallel backplane bus, or other Application_Processes in a unified manner.

A.2.2.6.3 Timing considerations

The worst-case transmission time from vehicle to vehicle is equal to the sum of the periods of the two Vehicle_Busses and of the Train_Bus. The worst case refresh delay is $T_{v1} + T_{v2} + T_t$, where T_x are the Basic_Periods of the Vehicle_Busses and the Train_Bus respectively. Therefore, the Train_Bus should exhibit a sufficiently high throughput and a deterministic response time.

A.2.2.6.4 Fault-tolerance considerations

The gateway gathers in one single Train_Bus frame (Dataset) the Process_Data of several devices located on its Vehicle_Bus. When a Vehicle_Bus device fails, the Process_Data it exports become invalid. If the device suffers only a partial failure, it may tag some of its data as invalid. If the whole device ceases to transmit, this is detected by the freshness supervision of the destination device, in this case the gateway.

When some of its input data are invalid, the gateway cannot just omit them from the frames it exports: the Train_Bus frames format may not change on-line. Therefore, the gateway tags the Process_Variables a failed device should have produced, as "invalid". The gateway overwrites suspicious fields with 0, regardless of the data

type. This ensures that the Check_Variables are overwritten with "0", so the Subscriber application can recognise the error.

The destination gateway forwards these invalid values over its Vehicle_Bus. The application in the destination device receives, instead of the value, the "invalid" indication and deals with it accordingly.

A.2.2.7 Application_Variables_Interface

The application interface is the only access the application programs should the Train Communication Network. This simplifies porting of applications from one type of device to another. The part of that interface which accesses the Variables service is called the Application_Variables_Interface, or AVI.

A.2.2.7.1 Individual Variables

Reading and writing Individual Variables

At the application interface, Process_Variables are accessed individually, although they may be part of a Dataset.

To update a variable it publishes, the Publisher application copies the value from its address space to the Traffic_Store through the procedure "ap_put".

To copy the value from the Traffic_Store to its address space, the Subscriber application calls the "ap_get" procedure as shown in Figure A.16.

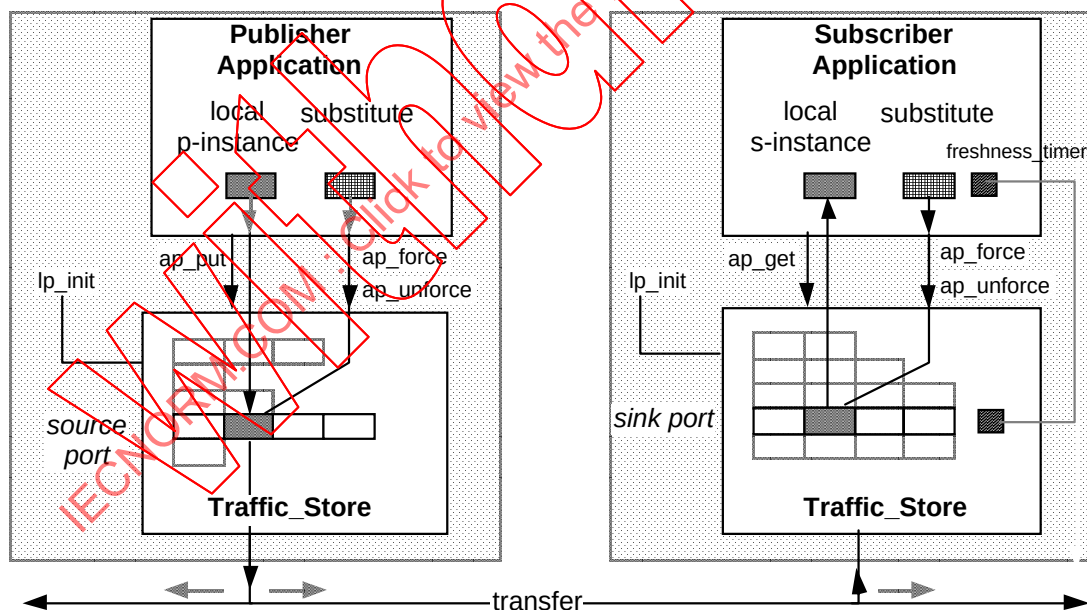


Figure A.16 - Individual copying

Copying respects the data presentation. For instance, if the Process_Variable is a boolean, it will be stored in the local instance in the representation the application expects, e.g. 0 for FALSE and -1 for TRUE. Another application could expect another representation.

Forcing Individual Variables

Differences between the instances may be introduced artificially for testing purposes. Any Application, but more likely the Network Management Agent, can force a Variable to a certain value. Forcing should be reserved to the Agent Function and should not be used during regular operation.

If the Application forces a Variable in a published Dataset, the forced value is distributed in place of the real value. This affects all Subscribers, since they cannot distinguish the real from the forced value.

If the Application forces a Variable in a subscribed Dataset (sink port), the forcing affects all Subscribers present on this device, but not on other devices.

If the variable is source and sink on the same device, all Subscribers are affected.

Forcing one Variable of a Dataset does not affect other Variables of the same Dataset.

The Application can indicate in the Check_Variable that the Process_Variable has been forced.

A.2.2.7.2 Variables Clustering

The individual access to a variable is time consuming. Reading of variables is preferably done once for all variables that a task requires at the beginning of that task's cycle, and writing at the end of the task's cycle. This is the preferred access mode for Programmable Logic Controllers using ladder diagrams or functional block languages. To this effect, a device holds Publisher and Subscriber cluster lists as shown in Figure A.17.

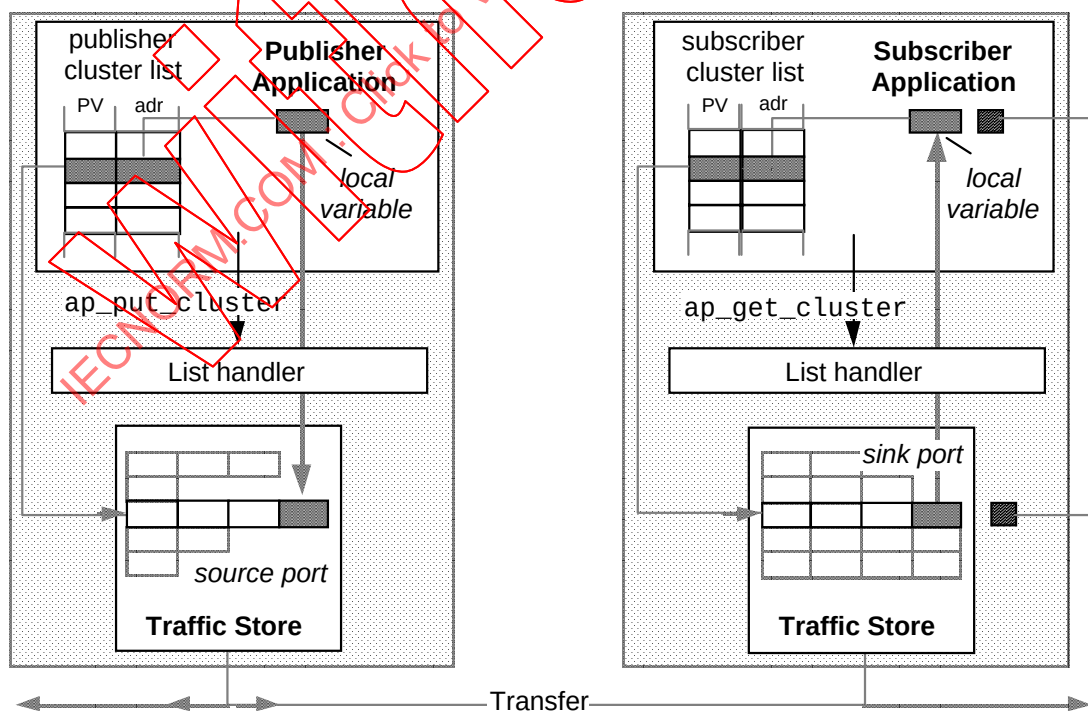


Figure A.17 - Cluster transfer

The Publisher application calls the procedure "ap_put_cluster" to request the transfer of a cluster of Process_Data from its own address space to the Traffic_Store. The

application specifies the variables and the local instances in the Publisher Cluster List. The list handler uses this list to access directly the address space of the Publisher.

A PV_Set is a subset of a cluster containing only variables of one and the same Dataset. This access service allows to consistent reading and writing of all variables of a Dataset.

A.2.2.7.3 Software stack for Process_Variables

Traffic_Store Access

Since Process_Variables are time-critical, an Application accesses the Traffic_Store, where Datasets are located, without involving an intermediate storage.

An Application accesses the Process_Variables through its Application_Variables_Interface (AVI), which provides the services of access to the Traffic_Store.

The AVI is implemented by the AVA (Application_Variables_Adapter), a software component which is linked directly with the Application program (it is part of the same process).

The AVA converts the data representation of the Application to the common bus format (and vice-versa). Since different Applications may use a different data representation, the AVA should be customised for each Application (and possibly each compiler if there are large differences).

The AVA accesses the Traffic_Store through the services of the Link_Process_Data_Interface (LPI). These services are provided by a software component, the Traffic_Store_Variables_Adapter (TVA), which is specific to an implementation of the Traffic_Store.

An Application accesses a Traffic_Store as shown in Figure A.18.

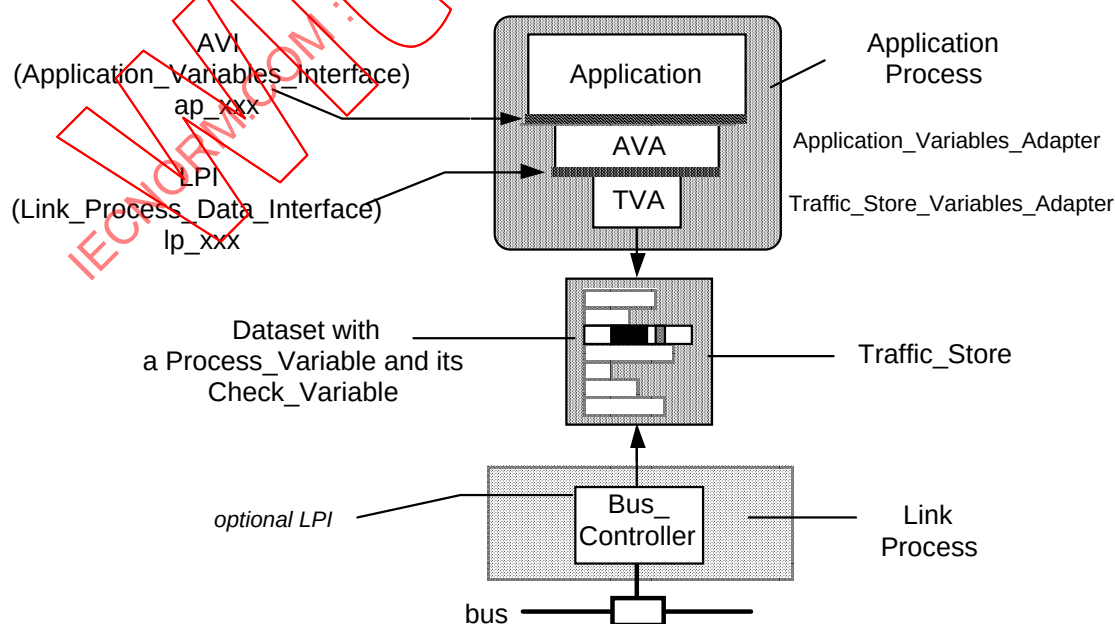


Figure A.18 - Application_Layer and Link_Layer Interface to Process_Variables

The Traffic_Store is controlled by the Link_Process, which conceptually runs in parallel with the process of the Application. The Link_Process may be implemented

in hardware by the Bus_Controller. The Link_Process may access the Traffic_Store through an interface similar to the LPI, but this interface is not standardised.

Multiple Applications

There may be several Applications running in parallel on the same device, which access the same Traffic_Store, as shown in Figure A.19.

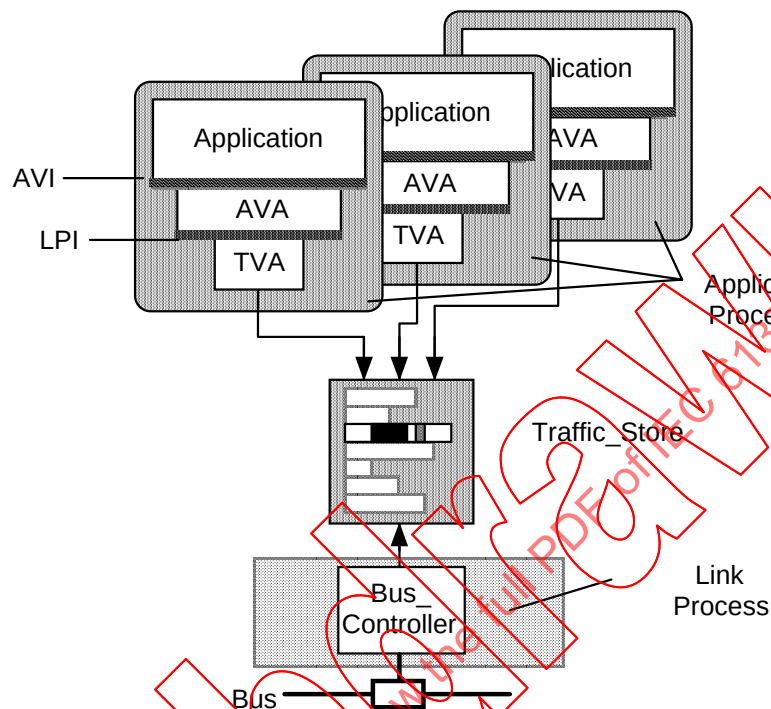


Figure A.19 - Several Applications access the same Traffic_Store

Applications may exchange data among themselves even if there is no network connection. In this case, the role of the Link_Process is reduced to the supervision of the freshness of the Process_Variables.

Although each Application may be at the same time Publisher or Subscriber, a specific Process_Variable may only have one Publisher.

Multiple Traffic Stores

There may be more than one Traffic_Store on a device. This allows a device to access several busses or to use Traffic_Stores for task-to-task communication within the same device.

Each Traffic_Store is attached to the AVA through its own TVA. The AVA chooses among different Traffic_Stores. The AVA and the TVA run always within the same process as the Application.

The bus accesses the Traffic_Store independently from the application. There is only one Link_Process for each Traffic_Store, as shown in Figure A.20.

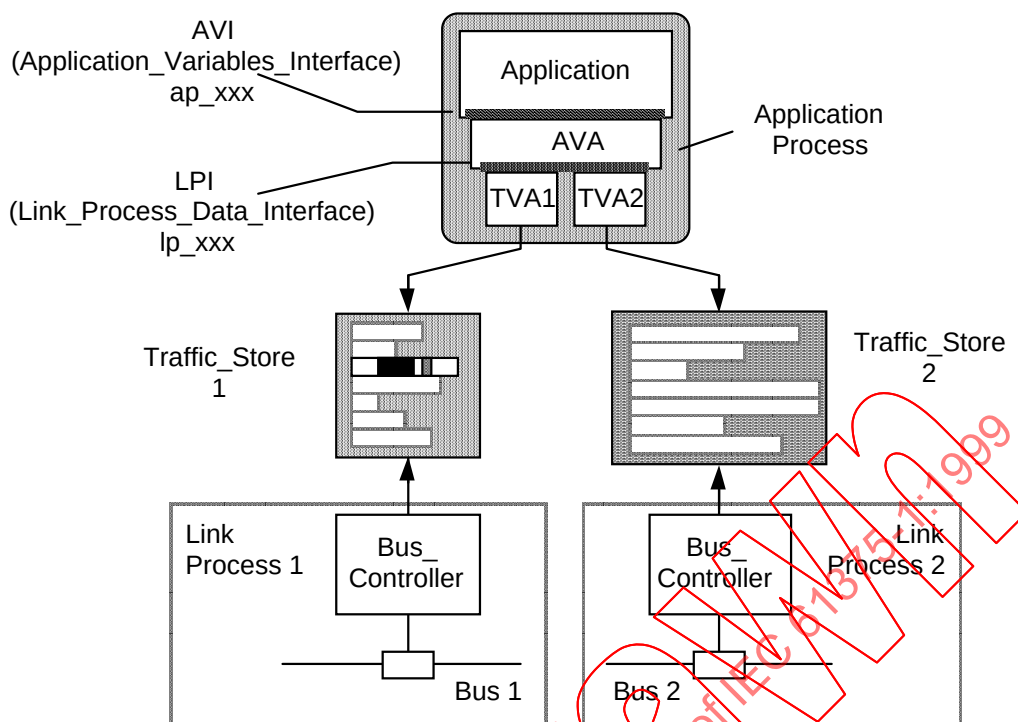


Figure A.20 - Application access to several Traffic_Stores

The Application_Variables_Interface AVI (between Application and AVA) is specified in this document. It is independent of the implementation of the Traffic_Store and of the Application.

The LPI is also specified in this document to ease interfacing with different busses.

This Standard does not impose a Traffic_Store structure.

A.2.3 Tutorial on the Messages Services

A.2.3.1 Reference architecture

The Train Communication Network follows a two-level hierarchy, the first level being the Train_Bus interconnecting vehicles and the second level being the Vehicle_Bus connecting devices within the vehicles as shown in Figure A.21.

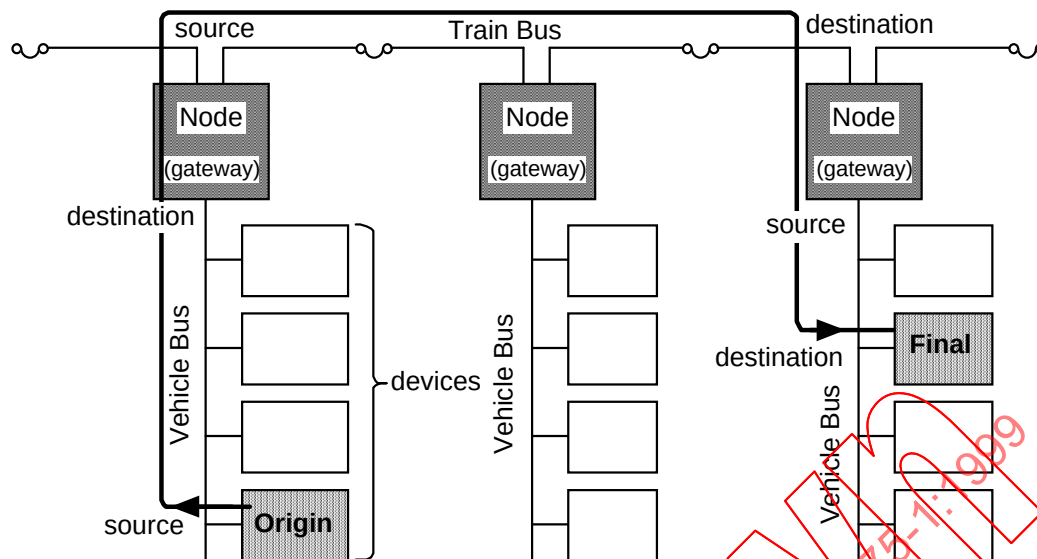


Figure A.21 - Two-level hierarchy

The Caller is the Application which initiates a message exchange by sending a Call_Message.

The Call_Message is forwarded over the network to another Application, the Replier, which is notified of its arrival.

The Caller is the Producer of the Call_Message, the Replier its Consumer.

The Replier receives the Call_Message and sends back a Reply_Message to the Caller, which is also notified of its arrival.

The Replier is the Producer of the Reply_Message, the Caller its Consumer.

A Conversation is composed of a Call_Message and the related Reply_Message.

A packet is sent by the Origin Station and received by the Final Station.

The data packets and control packets are exchanged using the Message_Data service of the Link_Layer.

Within the same bus, packets are transported by frames from a Source Device to a Destination Device.

Messages may also be transmitted using the Multicast Protocol. In this case, there is no Reply_Message.

System messages are Call_Messages or Reply_Messages exchanged exclusively between Managers and Agents for the purpose of network management. They identify each other by their System_Address.

User Messages are messages exchanged between User Applications. They identify each other by their User_Address.

NOTE Management messages are messages sent to an Agent for the purpose of network management. They may use a System_Address (network-wide) or a User_Address (within the same bus).

A.2.3.1.1 Communicating entities

The Messages Services allows an Application to transmit, on demand, data of any length, to another Application, located in another vehicle, in an other device or in the same device.

A device providing the Message Services is called a Station.

One Station may support several Applications. Within a Station, the same Application may exist in several copies, or Application instances.

Conceptually, all Applications of the TCN run in parallel, as separate processes (processes may be implemented as task, thread or similar constructs). For this reason, the term "Application_Process" is used to identify the end user of a message.

There exist two types of Application_Processes, which differ in the way they are addressed:

- The User_Processes execute Functions, which perform the actual work for which the control system exists, for instance doors control or anti-skid. Functions are not specified in this Standard, but users associations may standardise them in Companion Standards, such as UIC leaflet 556;
- Two Application_Processes are dedicated to Network Management:
 - the Agent present in each Station executes management functions, such as downloading, on behalf of a remote Manager. The agent is the only Application_Process present in every Station;
 - the Manager issues requests to be executed by the Agent on the same or on another Station, to which the Agent responds. The Manager is present only in a Management Station.

Managers usually communicate with Agents and User_Processes communicate with other User_Processes, but cross-communication is possible.

A.2.3.1.2 Messages

Applications use Messages rather than Variables to transmit large data items (such as diagnostic information or down-loading of devices).

Messages may also be used to transmit short commands (such as closing of a specific door), which would be too tedious to transmit periodically.

The Messages service relies on the ability of the busses to transmit, on demand, destination-oriented datagrams (Message_Data).

To ensure a reliable transmission from end to end, Applications communicate with each other on a Caller/Replier basis: The Caller Application sends a message to the Replier Application, which this one answers through a Reply_Message as shown in Figure A.22.

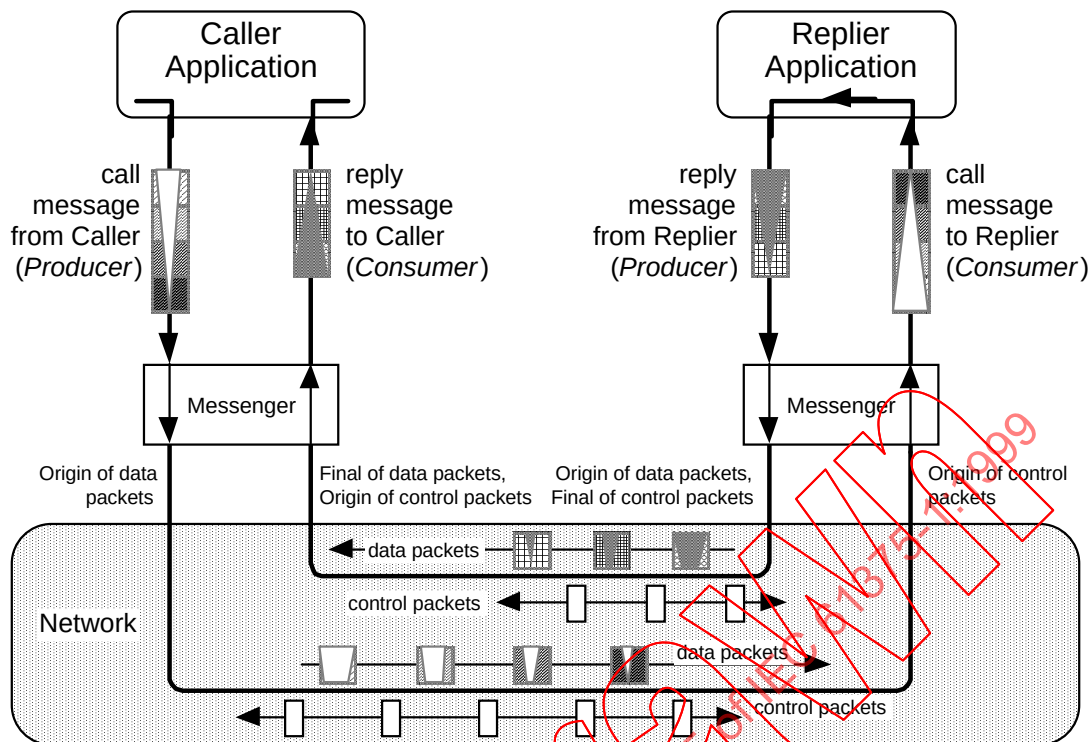


Figure A.22 - Call Message/Reply Message exchange

A.2.3.1.3 Messenger

The protocols are executed in each Station by the Messenger, which runs conceptually in parallel with the Applications, as a separate process.

A Caller sends a Call Message through the Application_Layer Interface to the Messenger.

The Session_Layer of the Messenger opens the connection (and closes it afterwards).

The Transport_Layer divides the message into a sequence of data packets small enough to fit into bus frames and sends them to the Network_Layer.

The Network_Layer consults its Function and Station directories to translate the addresses of the packets and forwards the packets to the Link_Layer.

The remote Messenger signals the arrival of a complete message to the Replier, which then can reply with a Reply_Message in the opposite direction.

The Message Transfer Protocol executed by the Transport_Layer of the Producer and of the Consumer performs flow control and error recovery to avoid loss or duplication of packets.

If both Applications reside on the same Station, the Session_Layer shortcuts the network and avoids segmentation. For instance, User Applications may require services of the local Agent by sending a message to it, without accessing the network.

The Messenger in each Station executes the transmission protocol. The Messenger at the Producer site divides the message into data packets, which the Messenger at the Consumer side acknowledges by means of control packets.

A.2.3.2 Actual architecture

The above two-level hierarchy is conceptual. An actual network may include Nodes with no Vehicle_Bus, or more than one Vehicle_Bus, include sensor busses and multiprocessors. Configurations in which no Train_Bus exists are also supported. The mapping of the conceptual network view and of the actual network is described in the following.

Figure A.23 shows a network consisting of a Train_Bus, the Node to the left having one Vehicle_Bus, the middle one having none and the one to the right having two Vehicle_Busses.

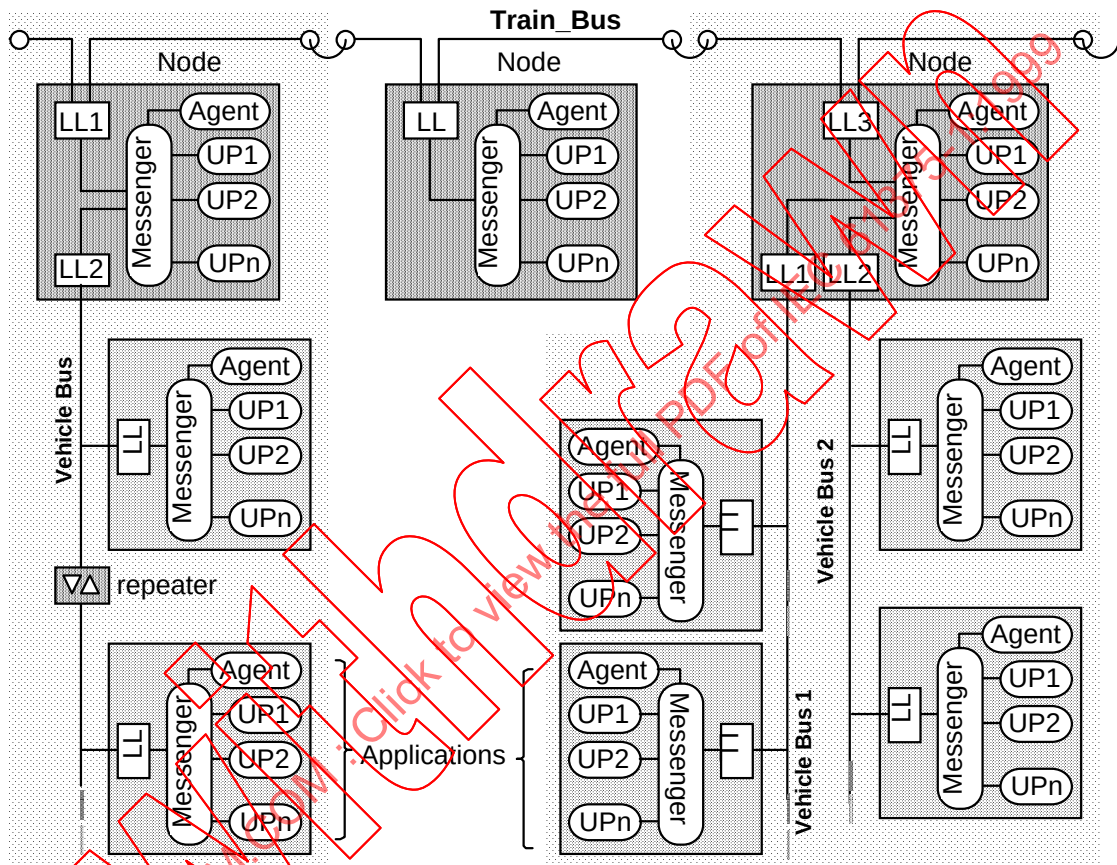


Figure A.23 - Example of actual architecture

Abbreviations: LL = Link_Layer UP = User Application

The presence of the repeater under the left Node is invisible to the protocols.

Terminal Stations have one Link_Layer, while Router Stations have more than one.

A.2.3.2.1 Message Data, Packets and Messages

Since messages could be too long to fit into one frame, they are segmented into packets, each packet fitting into one frame. These packets and the associated control packets (acknowledgements, commands, etc.) are carried as Message_Data over a particular bus.

The term "packet" applies therefore to the whole network (for instance, a packet is forwarded from vehicle to vehicle), while Message_Data is a service offered by a bus such as MVB or WTB to carry packets.

A.2.3.2.2 Message queues

Message_Data frames are transmitted from device to device within a bus. Each Message_Data frame carries the address of the destination device and of the source device.

Unlike Process_Data frames, a Message_Data frame may not update the preceding unread frame, since each one carries a unique piece of a larger information. Message_Data should be queued or processed before the next comes. If the preceding Message_Data could not be fetched, the incoming frame is discarded. To cope with load situations, the receiver device provides a Send_Queue and a Receive_Queue for Message_Data as shown in Figure A.24.

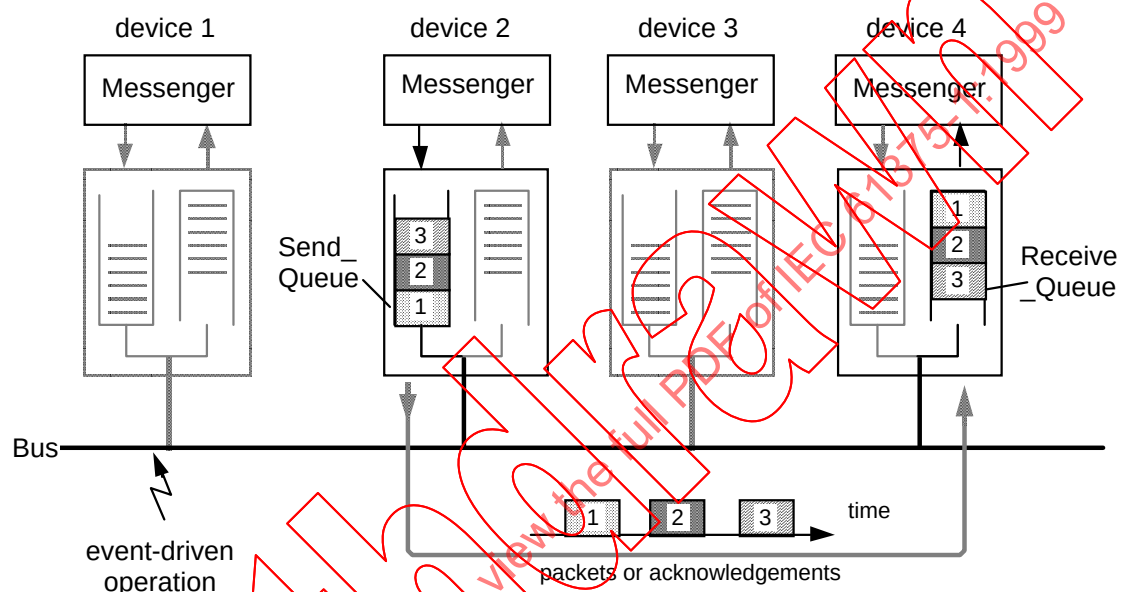


Figure A.24 - Message_Data transmission over queues

A.2.3.2.3 Message_Data medium access

Messages are transmitted on demand. Their response delay may vary depending on the network load. There is therefore no guaranteed delivery time for messages.

As for Process_Data, the Master polls the devices for Message_Data. A device responds to a poll with its Message_Data.

A.2.3.3 Messages Link Layer

A.2.3.3.1 Messages Link Layer protocol

The Link_Layer ensures transmission of frames between two devices belonging to the same bus. The Link_Layer itself is specific to the Train_Bus or the Vehicle_Bus, but the Link_Layer interface toward the higher (network) layer is the same for both busses.

Figure A.25 shown in the Link_Layer in the OSI hierarchy.

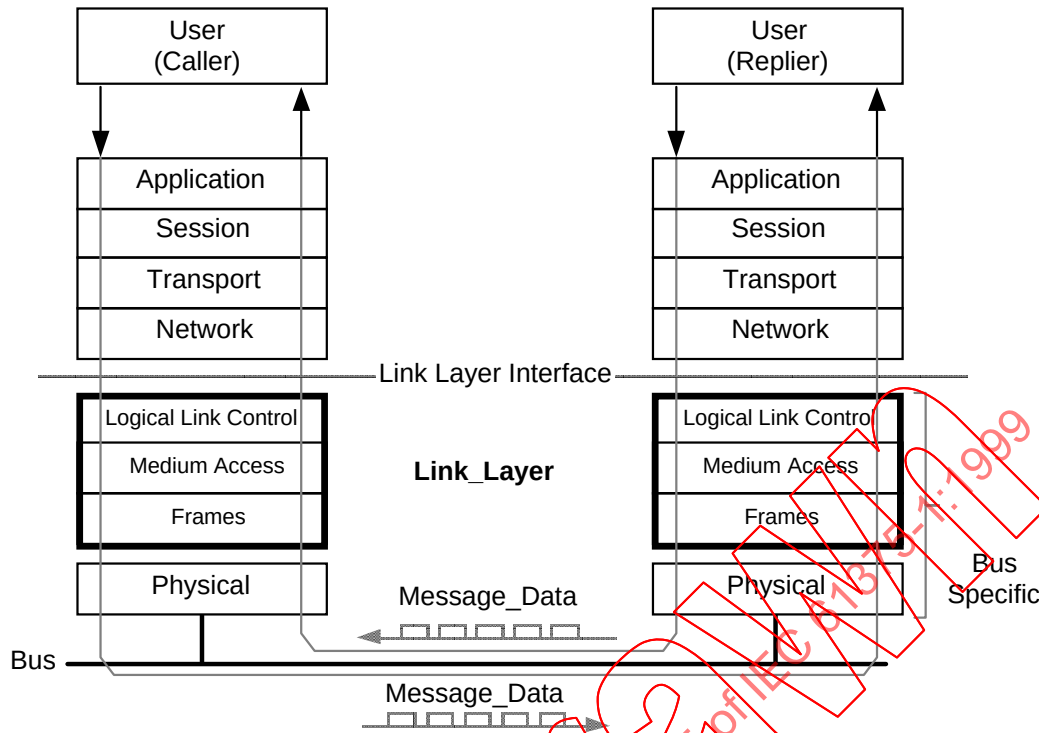


Figure A.25 - Link_Layer in the OSI hierarchy

The Link_Layer of the TCN is connectionless: there is no previous connection establishment, all Message_Data frames contain the full addresses required to transmit them from end to end. The Link_Layer does not control the flow neither does it recover lost frames, it just receives the frames on the basis of their destination address and queues them.

A.2.3.3.2 Message_Data format

Message_Data are transmitted as datagrams. A datagram is similar to a letter: each datagram carries all addresses needed to route it from end to end (and send the acknowledgements back). This scheme is advantageous when several busses are interconnected, since the routers do not need to keep knowledge about previous messages.

Each Message_Data frame carries two types of addresses: the source and destination Device_Addresses for the communication within one bus (Device_Addresses), and the addresses of the origin source and final destination (Network_Addresses).

The source and destination Device_Addresses apply only within the same bus. When Message_Data is sent to another vehicle, the destination device is the Train_Bus Node, which acts as router. When the Node receives Message_Data from another vehicle, it inserts its Device_Address as source device.

Message_Data carry two types of addresses:

- a source address and a destination address, which identify the devices communicating over the same bus and which are bus-specific. These addresses are specific to the Link_Layer;

- an Origin_Address and a Final_Address, which identify the Stations communicating over the network and which are understood in the whole network. The Origin_Address and the Final_Address identify the Producer and the Consumer, and in turn the Caller and the Replier. These addresses belong to the Network_Layer, which will be described below.

The frames on the WTB, the MVB or another bus only differ in their Link_Header as shown in Figure A.26.

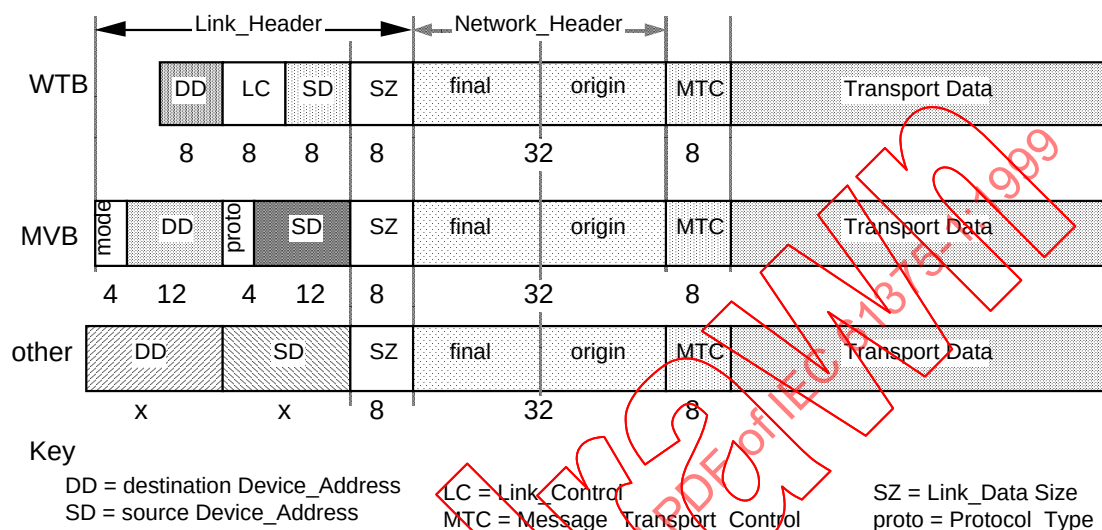


Figure A.26 - Message Data format

In all busses, a broadcast address may be used as destination address.

A.2.3.4 Messages Network Layer

A.2.3.4.1 Train Bus addressing

Railway operators use a variety of schemes for numbering vehicles within a train. The Train Communication Network provides an addressing scheme from which other address schemes can be deduced.

The TCN considers only Nodes, not vehicles. Its naming scheme allows a train to operate with arbitrary (and varying) travel direction and with a different number of Nodes per vehicle.

Each Node of the Train_Bus distinguishes two directions, named "Direction_1" and "Direction_2".

A vehicle is wired in such a way that Direction_1 of a Node corresponds to Direction_1 of its vehicle. If a vehicle has several Nodes, they all have the same Direction_1.

NOTE UIC defines Direction_1 as the extremity of the vehicle where the parking brake is located.

The two sides of the vehicle are called Side_A and Side_B. The orientation of Side_A and Side_B with respect to the Direction_1 and Direction_2 of a vehicle is defined such that if Direction_1 is heading north, Side_A is heading west.

Irrespective of the travel direction, the Train_Bus identifies Nodes by their position relative to the Master Node, whose address is always 01 as shown in Figure A.27.

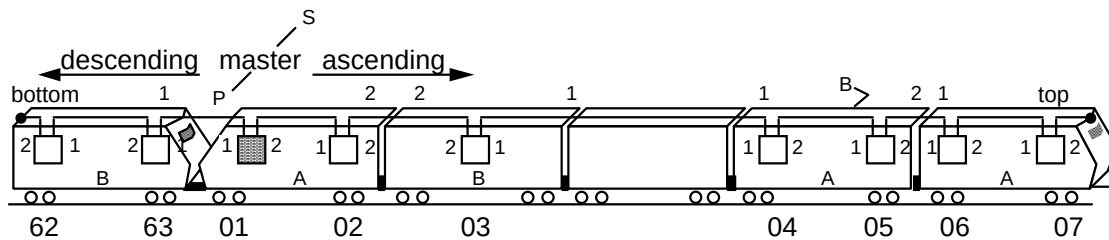


Figure A.27 - Node position numbering

Nodes in Direction_2 of the Master are sequentially numbered in ascending order starting with 02, the last named Node being the top Node.

Nodes in Direction_1 of the Master are sequentially numbered in descending order, starting with 63, the last named Node being the bottom Node.

Each Node knows its address, which of Direction_1 or Direction_2 points to the Master and whether it is on the bottom or on top of the Master. For instance, in Figure A.7, Node 03 will consider its Direction_2 as "bottom" and Node 05 its Direction_2 as "top".

Thus, all Nodes recognise the same direction as "top" or "bottom", and know whether their vehicle has the same or the reverse orientation as the vehicle where the Master resides. The side of a vehicle which is on Side_A of the vehicle where the Master resides is called "P", the opposite side "S".

Thus, if the application commands the opening of doors on the side P (corresponding to side "A" of the Master), all Nodes with the same orientation as the Master will open their "A" doors, and the other Nodes will open their "B" doors, without knowing the direction of travel.

Mapping of Node addresses to application addresses

Railways operators identify vehicles within a train. rather than Nodes within a bus.

The convention for numbering vehicles varies from operator to operator. In most cases, the crew identifies a vehicle by its position with respect to the front of the train, for instance according to the convention specified in the UIC leaflet 556 as shown in Figure A.28.

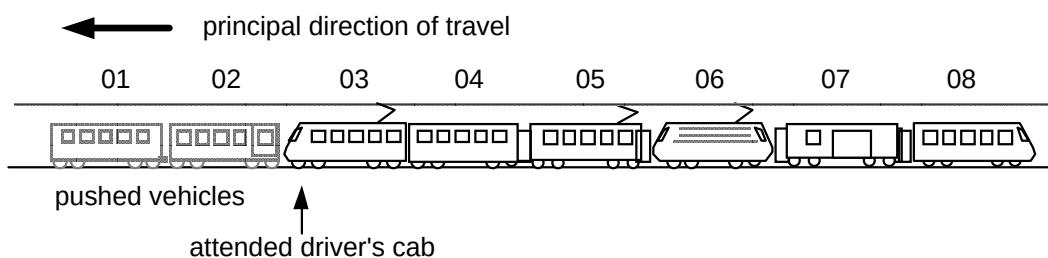


Figure A.28 - Vehicle numbering according to UIC 556 - not TCN

In UIC 556, the front vehicle is identified as vehicle number 01. It is normally the vehicle where the driver is located, but there may be pushed vehicles.

The vehicles are numbered sequentially towards the rear of the train. Vehicle identifiers do not change when the train drives backward, as long as the driver remains in the same location.

Other operators number only passenger coaches, possibly excluding postal coaches, others number axles rather than vehicles.

The seat reservation assigns a 3-digit number to a coach, independently from its position - a coach keeps its reservation number when it changes train.

To add to this diversity of addressing schemes, the UIC identifies each coach by a unique, 15-digit identifier and the manufacturer possibly by a serial number.

In addition, it may be necessary to correct the configuration to consider continuity vehicles or vehicles with disabled equipment.

Indeed, it is not sufficient just to compensate for missing Node_Addresses. Since vehicles without operative Nodes become invisible to the TCN, their characteristics should be entered by other means. Otherwise, parameters such as braking distances and platform position would be affected.

Therefore, a mapping module in each device translates the application addresses (UIC or seat reservation number) to TCN addresses and vice-versa to suit the addressing scheme of each application.

Node_Descriptor and Vehicle_Descriptor

To ensure that all Nodes share a consistent view of the bus, all nodes should possess a copy of the Topography, which indicates the number, position and Node_Descriptor of all Nodes.

On the WTB, the Master sends the topography at each change of composition to each node before going to regular operation.

In fixed compositions, the Topography can be stored in each node by using Network Management services and it should not be modified during regular operation.

The Topography contains the 16-bit Node_Descriptor, which indicates which type of Process Data the Node publishes and which Functions it supports. Its content is defined by a particular application and is outside the scope of this Standard.

In simple cases, the Node_Descriptor can be used to identify types of vehicles (street cars, for instance). The general case is more complicated, since:

- there may be more than one Node in a vehicle, particularly in locomotives;
- there may be vehicles without a Train_Bus Node (continuity vehicles);
- some Train_Bus Nodes may be inoperative;
- in suburban trains or metros, the Vehicle_Bus may span several vehicles which share a single Train_Bus Node.

In these cases, the application may use the Node_Address and Node_Descriptor to ask each Node for its Vehicle_Descriptor, in which more detailed information is available. The Vehicle_Descriptor is subject to other standards or agreements between manufacturers and users. For instance, the UIC 556 leaflet specifies a Vehicle_Descriptor, which contains the seat reservation identifier, the UIC number, the vehicle length, weight, maximum speed, etc.

Group_Address

To access vehicles by their category or groups of vehicles, a Group_Address may be specified instead of a Node_Address. The assignment of vehicle categories to groups is application-dependent. Examples of vehicles as shown in Figure A.29.

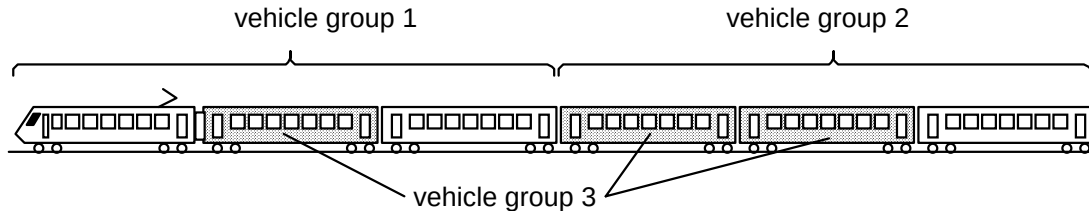


Figure A.29 - Example of vehicle types

Addressing of groups requires that packets are sent in broadcast over the Train_Bus. The Group_Address is contained within the final Network_Address.

To this effect, each Node holds a Group_Directory, which indicates to which groups the Node pertains.

Mapping of Node addresses to device addresses

When the WTB is used as a Train_Bus, the inauguration assigns to each node a Device_Address which is equal to the Node_Address.

Therefore, the 6-bit Node_Address is mapped directly to the 8-bit WTB Device_Address by setting the two most significant bits of the Device_Address to 0.

Applications which do not require an inauguration (e.g. fixed consists) may use other busses than the WTB, for instance the MVB. In this case, the devices can be assigned a sequential address during commissioning, allowing a one-to-one mapping between the 6-bit node address and, in the case of the MVB, the 12-bit Device_Address.

A Node_Directory, which maps the node address to a device address of any size, provides a flexible scheme for applications where changing the Device_Address is cumbersome.

A.2.3.4.2 System view

Station_Identifier

When commissioning or debugging, the system engineer is interested in accessing devices through Management_Messages. Due to the variety of configurations, it is not possible to use physical device addresses as a general scheme.

In the example of Figure A.30, vehicle 02 has no Vehicle_Bus, while vehicle 05 has several Vehicle_Busses and vehicle 03 has two Train_Bus Nodes which can be accessed separately. Finally, sensor busses can also be present.

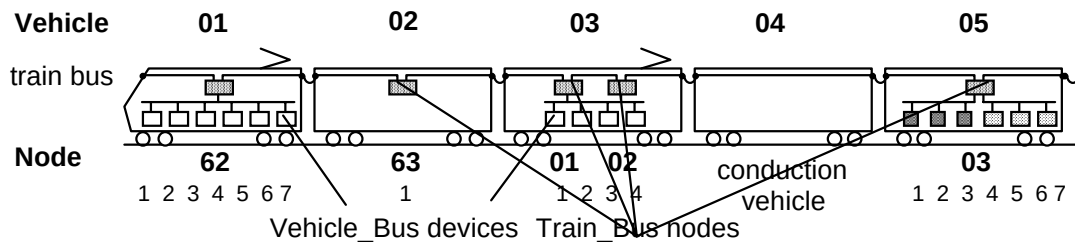


Figure A.30 - Nodes and Vehicle_Bus Devices

A Vehicle_Bus can also span a set of vehicles which may not be separated in normal operation, and each set may be connected to one Node of the Train_Bus, as shown in Figure A.31.

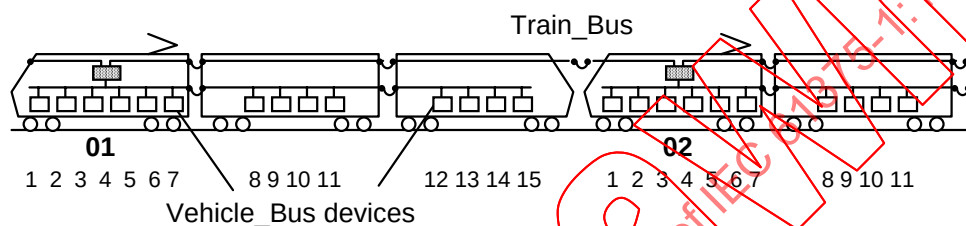


Figure A.31 - Vehicle_Bus spanning several vehicles

To address devices under various topologies, Network_Management considers the Train Communication Network as a number of Stations attached to each Node, the Node itself being one of them as shown in Figure A.32.

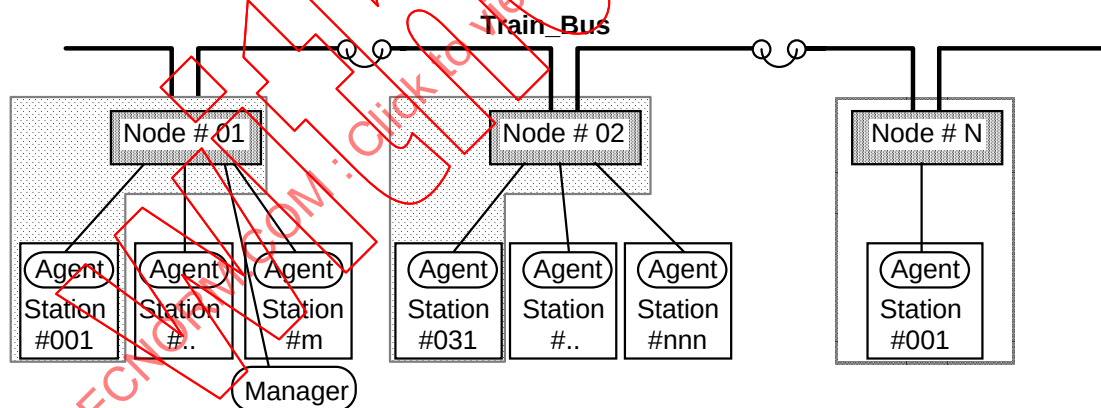


Figure A.32 - System view of communication

To Network Management, only Stations (devices capable of message communication) are visible. Each Station is identified by its Station_Identifier.

There may be up to 254 Stations attached to a Node.

System_Address

The Manager identifies an Agent by its Node_Address and Station_Identifier. It considers the Node as a Station as any other (not necessarily Station #001).

The Manager can access jointly several Stations (on different Nodes) by using a Group_Address instead of a Node_Address .

For special purposes, the Manager and the Agent can specify a Router Station, identified by its Next_Station_Identifier, which can forward packets to the intended Station. Within the same bus, this Next_Station_Identifier serves to access a specific Device.

Stations are accessed by their System_Address, which consists of a Node_Address (or a Group_Address), a Station_Identifier. and an optional Next_Station_Identifier.

Station_Directory

The Manager accesses the devices by their Station_Identifier, rather than by their Device_Address. This makes its view independent of the addressing scheme of a particular bus and of the number of Vehicle_Busses.

Therefore, there should be a one-to-one mapping between the Station_Identifier and the Link_Address of a device (Bus_Id and Device_Address).

A Station_Directory provides a general mapping, which for each Station_Identifier indicates the corresponding Bus_Id (bus) and the Device_Address on the bus as shown in Figure A.33.

		Link_Address	
Final_Station	Next_Station	Bus_Id	Device_Address
#003		MVB1	0003
005	-	MVB1	0005
054	103	MVB2	0003
103	-	MVB2	0003
223	-	Parallel_Bus	203040

Figure A.33 - Station_Directory

Each Application can in principle access the Station_Directory, but it is expected that the Agent will be the Application in charge of managing the Station_Directory.

Static entries in the Station_Directory are made at configuration time.

An Application may make changes to the Station_Directory, which apply to all other Applications. Changing the Station_Directory during regular operation bears the risk of disrupting or wrongly delivering messages.

Simple Routing

When the Link_Address can be derived directly from the Station_Identifier, the Station_Directory may be implemented by an algorithm rather than a table. This is called simple routing.

For instance, on MVB, the 12-bit Device_Address can be directly derived from the 8-bit Station_Identifier by extending the most significant four bits with zeros. This requires that the Station_Identifier be derived from the Device_Address. Conversely, the Station_Identifier is deduced by stripping off the most significant four bits (if they are zero).

The same scheme may be used when several Vehicle_Busses are connected to the same router. In this case, the number of devices per bus is reduced.

For instance, Stations 1..31 may be assigned to bus 1, and Stations 32..127 to bus 2.

Simple routing limits the degree of freedom of the structure of the Vehicle_Busses, but is more efficient than Station_Directory routing.

Example: Addressing with a System_Address

NOTE - In the following text and figures:

- a four-digit number represents the Device_Address of a Vehicle_Bus device;
- a three-digit number preceded by a # sign represents a Station_Identifier;
- a two-digit number represents the Node_Address of a Node;
- a one-digit number represents the Device_Address of a sensor bus device.

In the example of Figure A.34, Station #004 attached to Node 05 sends a packet to Station #125 attached to Node 07.

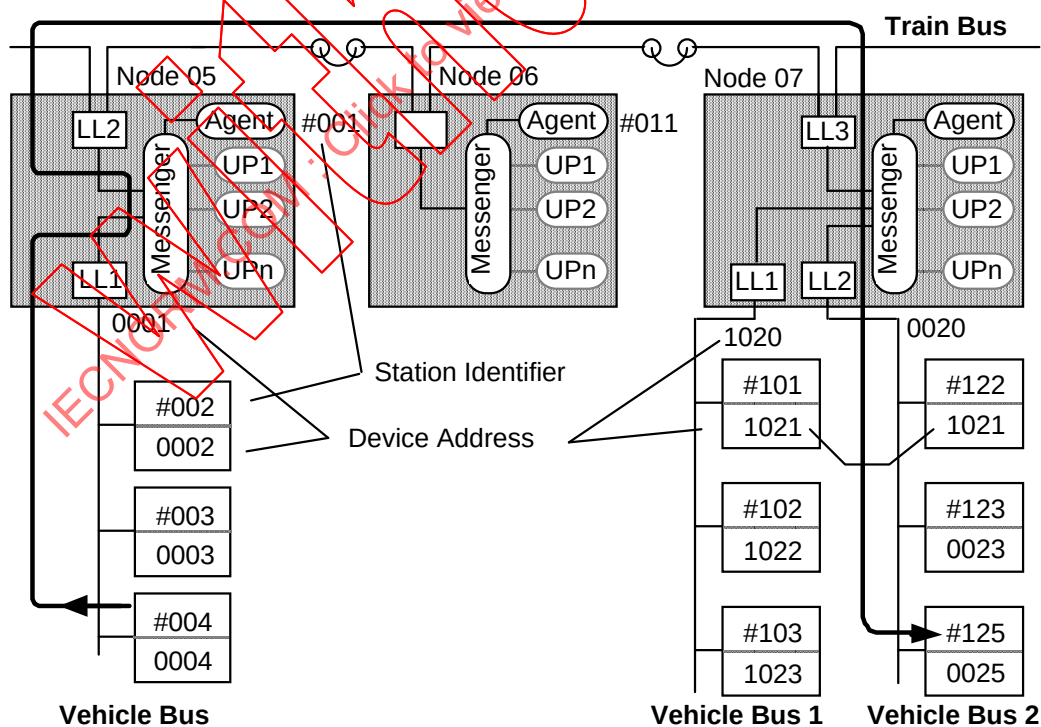


Figure A.34 - Actual Station location

When Node 07 receives a packet intended for Station #125, it deduces from its Station_Directory that Station #125 is located on Vehicle_Bus #2 (Bus_Id = LL2) with Device_Address 0025.

It is legal to have two devices with the same Device_Address (e.g. 1021) on different busses attached to the same Node, however, they should have a different Station_Identifier. The router distinguishes them by their Bus_Id (LL1 or LL2).

Example: Multiprocessor Node

The Station_Directory allows the introduction of sub-levels in the hierarchy. As an example, shown in Figure A.35 the routing of packets in a dual-processor Node.

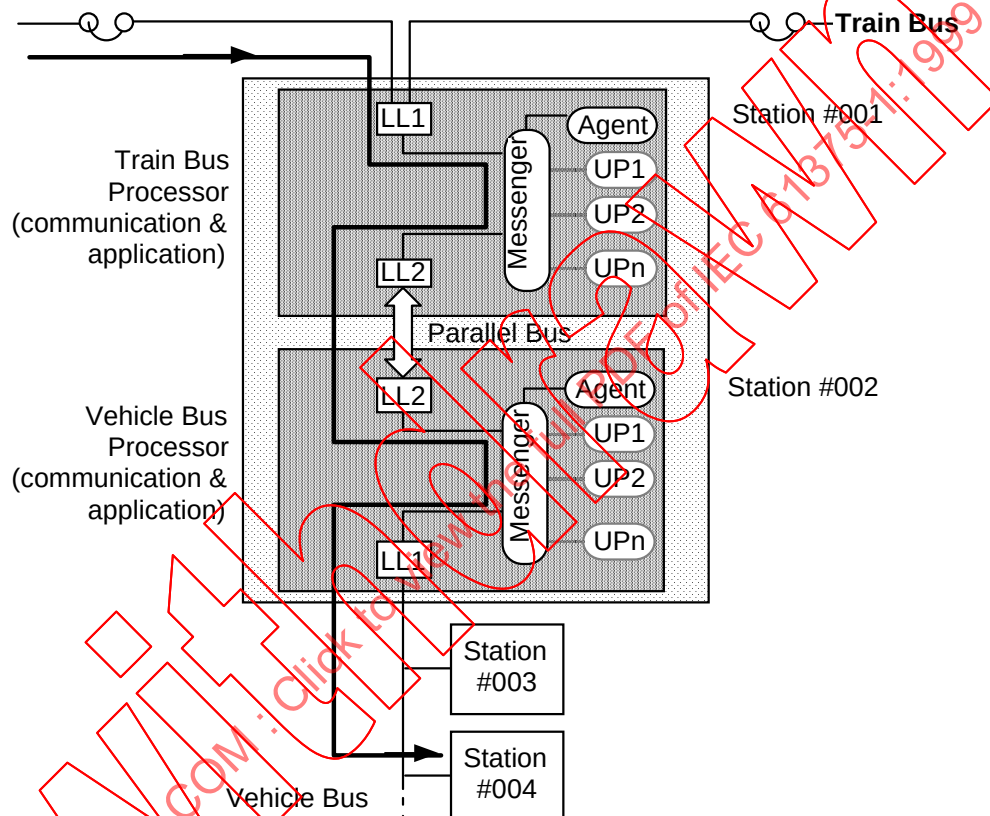


Figure A.35 - Dual-processor Node

One processor is dedicated to the Train_Bus traffic, while the other is dedicated to the Vehicle_Bus traffic. Both can run Applications and both have a Messenger.

Each processor is a Station, with its own Station_Identifier and Agent. When the Train_Bus processor (Station #001) receives a message intended for Station #002, it checks its Station_Directory as to which Link_Address to access Station #002, and it forwards the packet over its Bus_Id = LL2 (which accesses a parallel bus).

Conversely, Station #002 is a router for all Vehicle_Bus devices connected to it and forwards packets intended for the Train_Bus over its Link_Layer = LL2.

Example: Sensor bus

The Station_Directory allows the introduction of a further level of hierarchy, for instance a sensor bus. Sensor bus devices are simple devices, which generally support only an Agent.

In the example of Figure A.36, the origin Station #133 attached to Node 05 sends a packet to final Station #209 attached to Node 08 over Station #106.

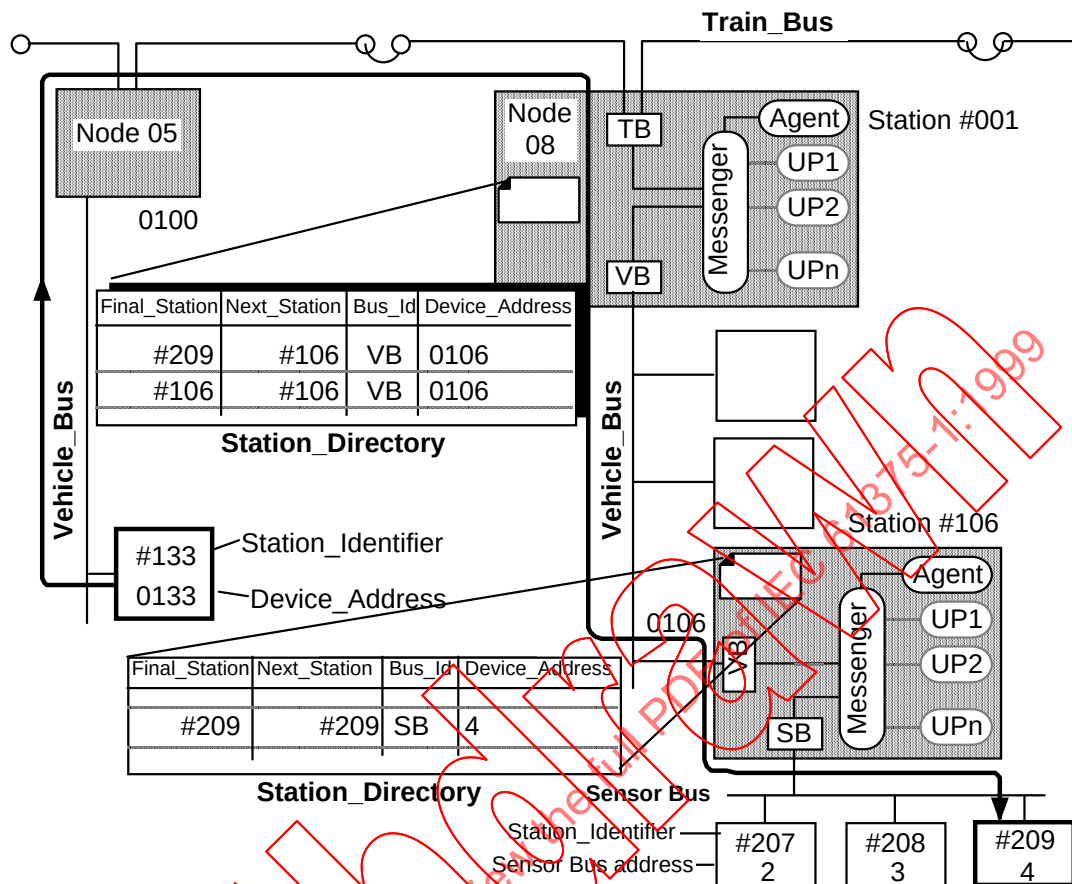


Figure A.36 - Further hierarchical level

SB = Sensor_Bus Link_Layer, TB = Train_Bus Link_Layer, VB = Vehicle_Bus Link_Layer

In the Node, all sensor bus devices appear under the Station_Identifier of the device to which the sensor bus is attached (Station #106 in the case illustrated in figure A.40). The router on Station #106 continues the routing and forwards the packet to the sensor bus device.

A.2.3.4.3 User view

Functions

Each vehicle supports a number of Application Functions, as in the example as shown in Figure A.37.

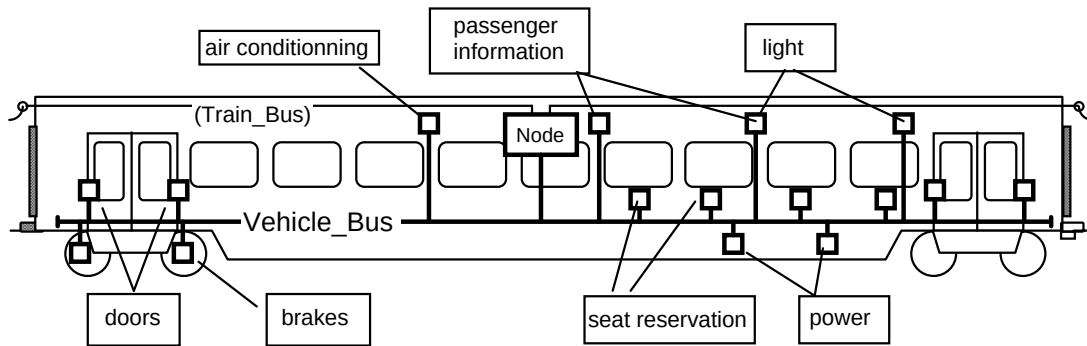


Figure A.37 - Functions within a passenger coach

For instance, the UIC 556 leaflet defines a number of Functions supported in all compliant UIC trains. Within a closed system (metro, for instance), Functions are agreed between the manufacturer and operator.

The Functions are executed by devices. The mapping of Functions to devices depends on the implementation. A Function may be implemented by a device connected to the Vehicle_Bus or directly by the Train_Bus Node.

One device may implement several Functions, or conversely, a Function may be implemented by several devices. For instance, there may be two devices which participate in the "doors" Function as shown in Figure A.38.

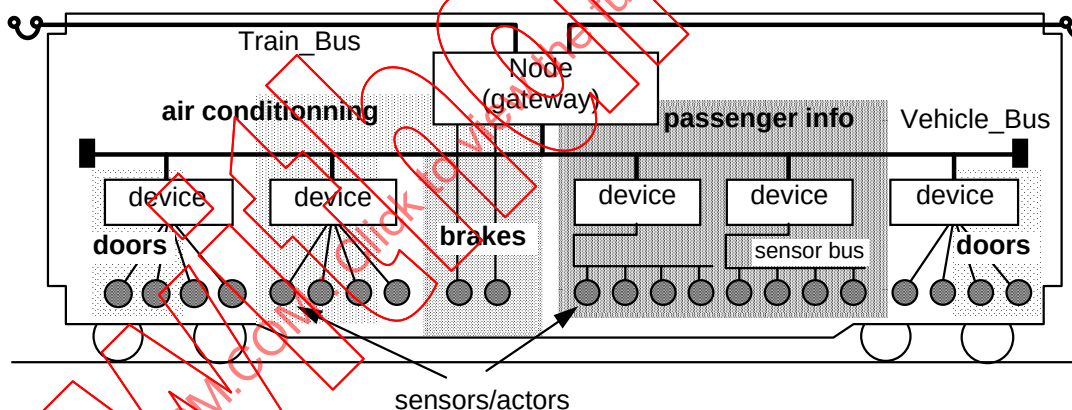


Figure A.38 - Mapping of Functions to Devices

Vehicles of different nationality and age may be coupled in the same train. Due to the diversity of vehicle implementations, one can assume no knowledge of Device_Addresses at the Train_Bus level. To access the air conditioning on another vehicle, a vehicle would should know the Device_Address of all possible vehicle devices, which is unrealistic.

The Train_Bus user is not expected to know which devices a vehicle contains. It considers that each vehicle supports a number of Functions (doors, brakes, etc.), but does not specify which device executes a specific Function.

For this reason, a user does not address a remote device, but a remote Function, by specifying the Node to which this Function is attached and the Function_Identifier.

The Functions may be implemented in the Node itself or in any device connected to the Node. The Node routes the messages to the correct device.

Addressing through Functions rather than Devices also applies within a Vehicle_Bus: An Application calls a Function without specifying which device executes it.

User Applications see the network as consisting of a number of Nodes, each Node supporting a number of Functions, as shown in Figure A.39.

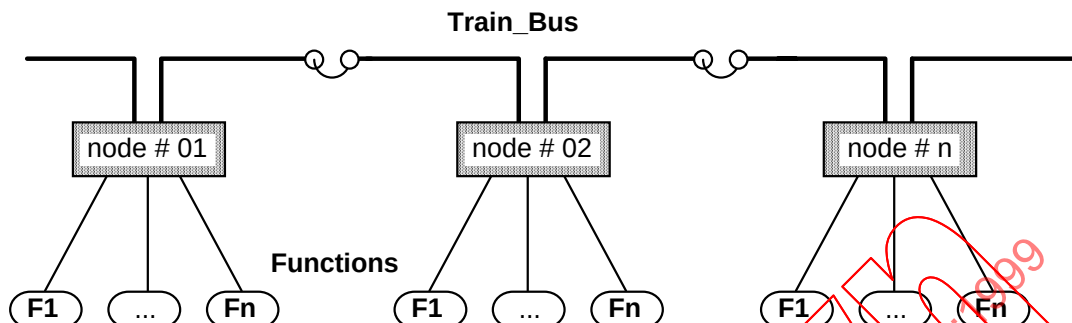


Figure A.39 - Function view of communication

The User Application normally does not specify which station executes the Function.

However, a Caller or a Replier may specify in the User_Address the Next_Station over which the Function can be reached.

Functions are accesses by their User_Address, consists of the concatenation of a Node_Address (or a Group_Address), a Function_Identifier and an optional Next_Station_Identifier.

Function_Directory

For each registered Function, the Function_Directory indicates which Station executes the Function. The Station_Identifier of this Station is called Final_Station.

Final_Station is in turn resolved by the Station_Directory into Next_Station, Bus_Id and Device_Address.

Next_Station indicates over which Station the Final_Station can be reached if it is not directly accessible, as shown in Figure A.40.

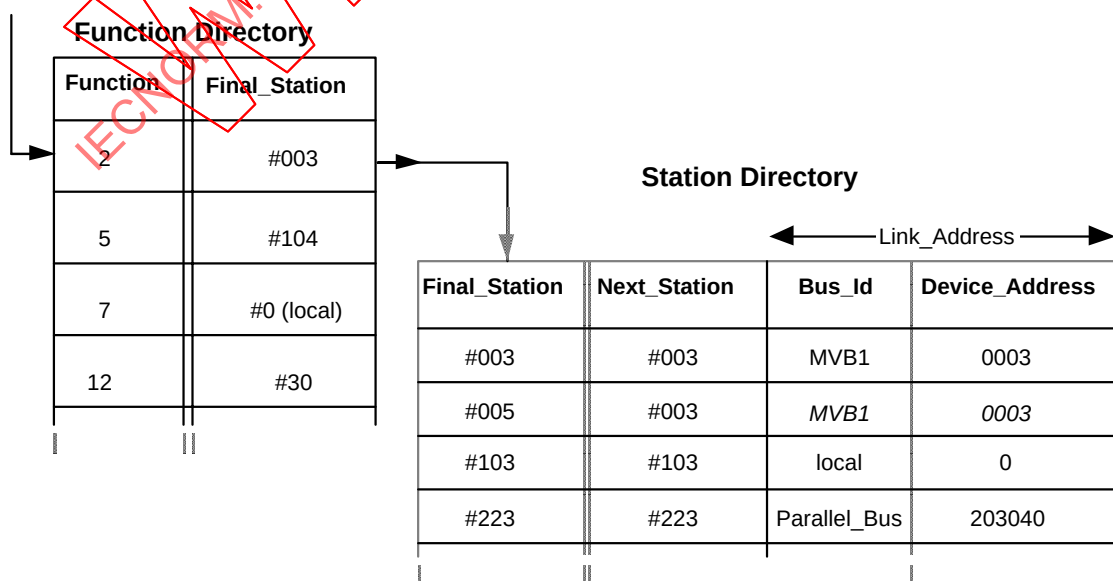


Figure A.40 - Function_Directory

A Function_Directory is needed for each Messenger even if a device has no bus connection, since it is also used to route messages within the same Station.

The Session_Layer accesses the Function_Directory to check where the partner resides, and if it resides on the same Station, it transmits the message directly.

Example: Addressing with User_Address

Figure A.41 illustrates the location of the Function Directories.

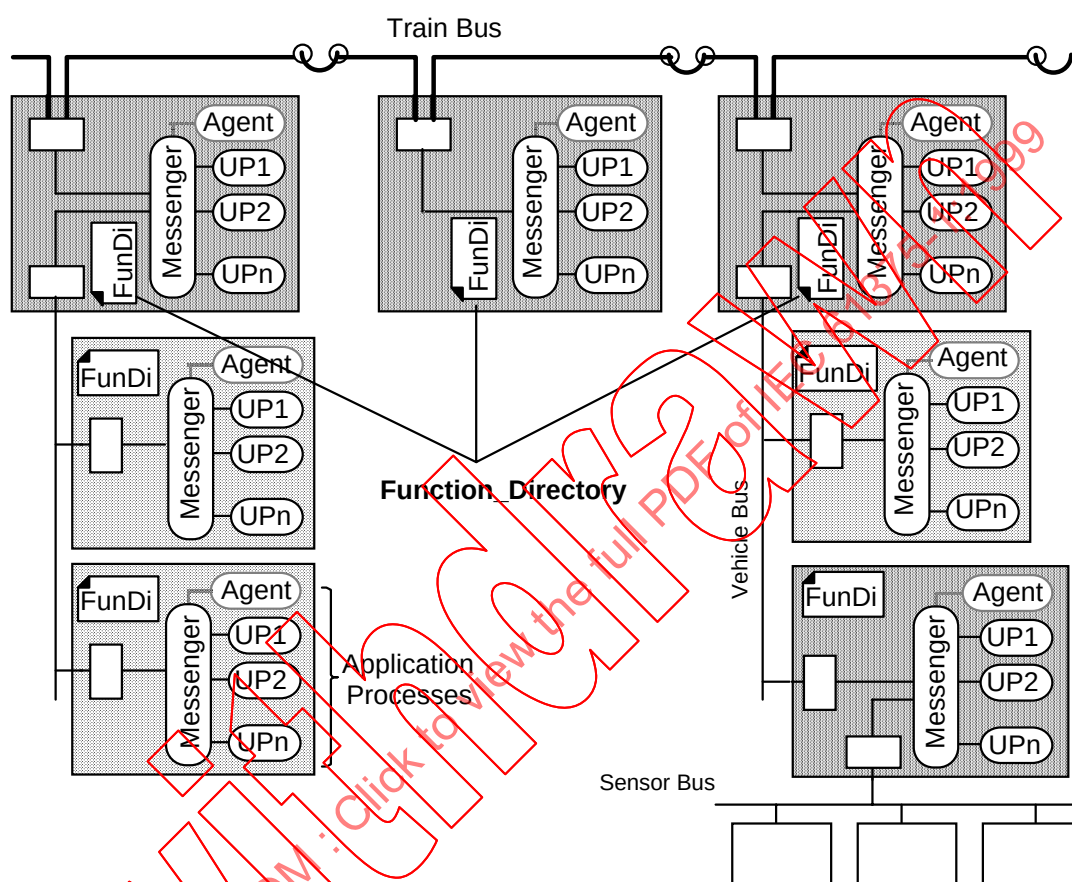


Figure A.41 - Function Directories in a three-level hierarchy

Even Nodes without a Vehicle_Bus, such as the centre Node in Figure A.41, require a Function_Directory for node-internal communication.

Homonym Functions

A Function existing on several devices under the same Node is called a Homonym Function. This situation is likely to occur when, for instance, there are several identical devices on the same bus.

Homonym Functions are distinguished using their Next_Station_Identifier, which specifies the Link_Address (Bus_Id and Device_Address) to which the packets should be forwarded to.

When the user specifies Next_Station in the Replier address, the Function_Directory is by-passed.

The Node should make a choice between the homonym Functions when a message is received for one over the Train_Bus.

A Function_Directory has only one entry for each Function. The Station corresponding to this entry is the Function Leader. This Station behaves as if it would implement the Function by itself. To do so, it may access the other devices implementing the homonym Functions, or Function Followers by specifying Next_Station. The Function Leader is not necessarily located on the device which implements the Function: it may also be another device or the Node itself.

A.2.3.4.4 Packet routing

The Train Communication Network transmits Packets (data, acknowledgements or control frames for messages) from bus to bus without previous connection: a router keeps no knowledge about former Message_Data.

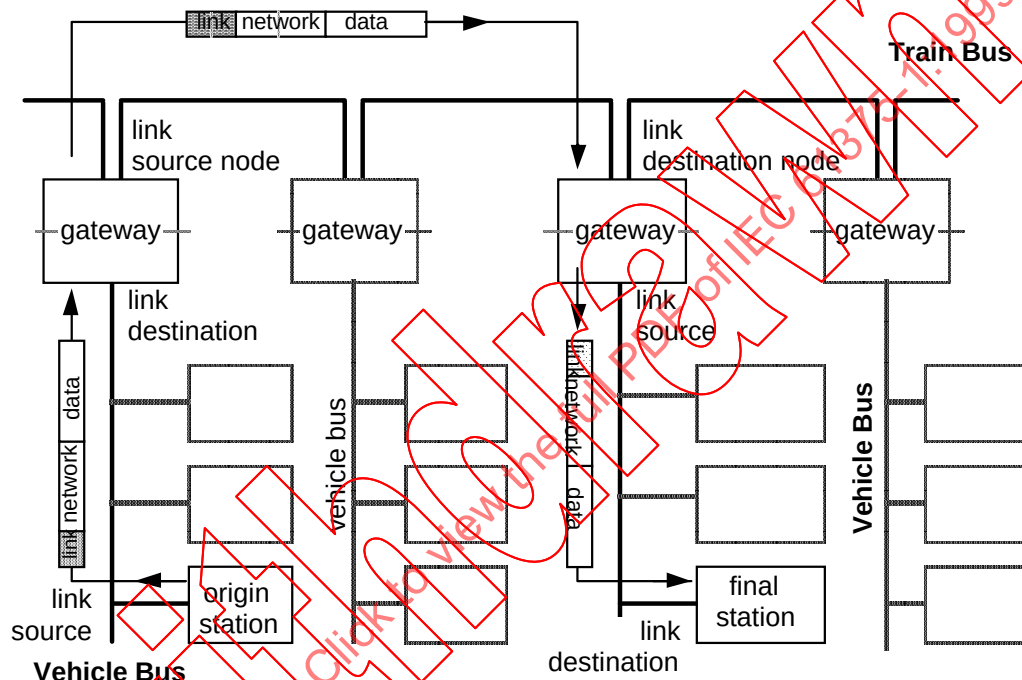


Figure A.42 - End-to-end Message_Data transfer

The Function which issues the Message_Data resides in the origin Station. The Function which uses the Message_Data resides in the final Station (see Figure A.42). When both Functions reside on the same bus, the source Station is the origin Station, and the destination Station is the final Station. The origin or destination Station may also be a Train_Bus Node.

A Train_Bus Node is a router for Message_Data. It has two interfaces, one towards the Vehicle_Bus and one towards the Train_Bus, as the OSI model illustrate in Figure A.43.

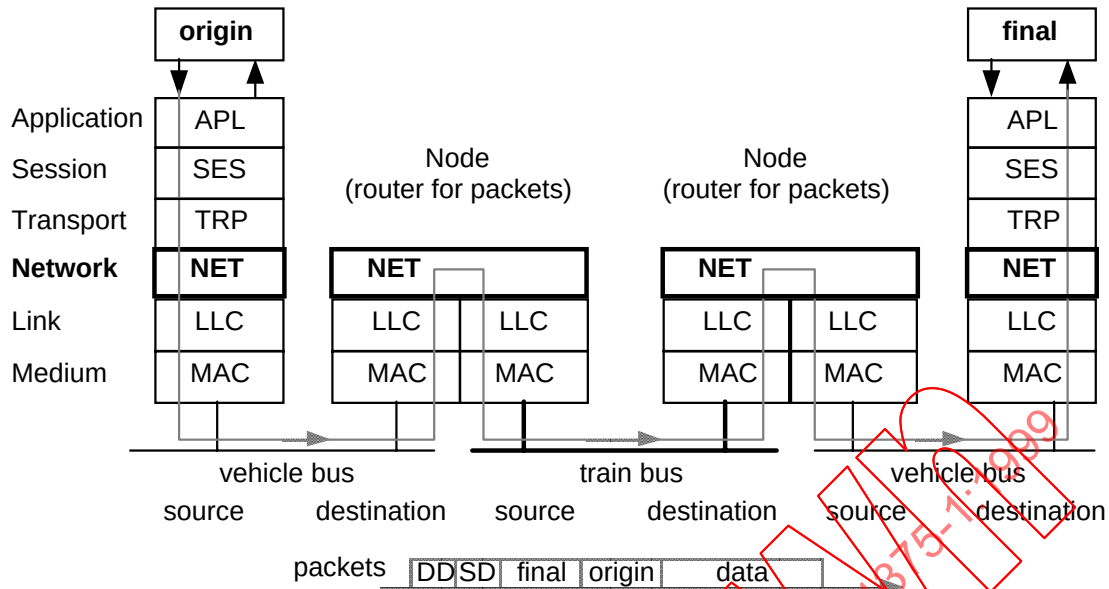


Figure A.43 - Packet forwarding over the network

Network_Address

The Train Communication Network uses a global, hierarchical address for messages. There are two types of Network_Addresses, which are both 16 bits long:

- the User_Address, which is used for identifying Functions; It consist of a 7-bit Node_Address or Group_Address (e.g. all traction vehicles) and an 8-bit Function_Identifier;
- the System_Address, which is used for identifying Stations for Network_Management; It consists of a 7-bit physical Node_Address and an 8-bit Station_Identifier as shown in Figure A.44.

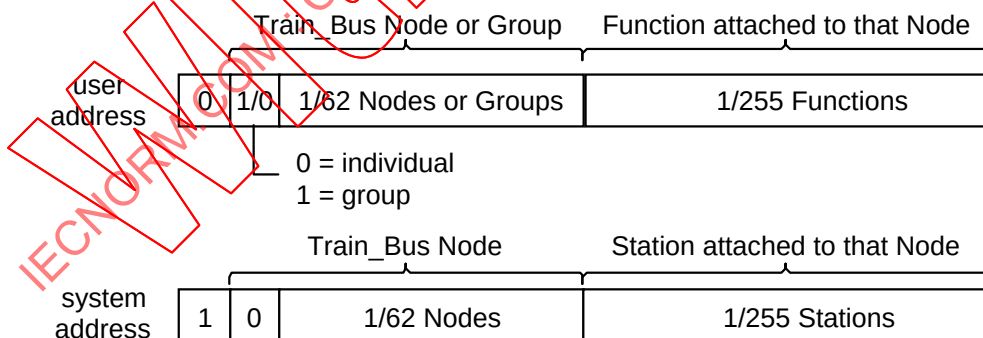


Figure A.44 - Network_Address (origin or final) for messages

Each Message_Data frame bears both the Network_Address of its final destination and the Network_Address of its origin as shown in Figure A.45.

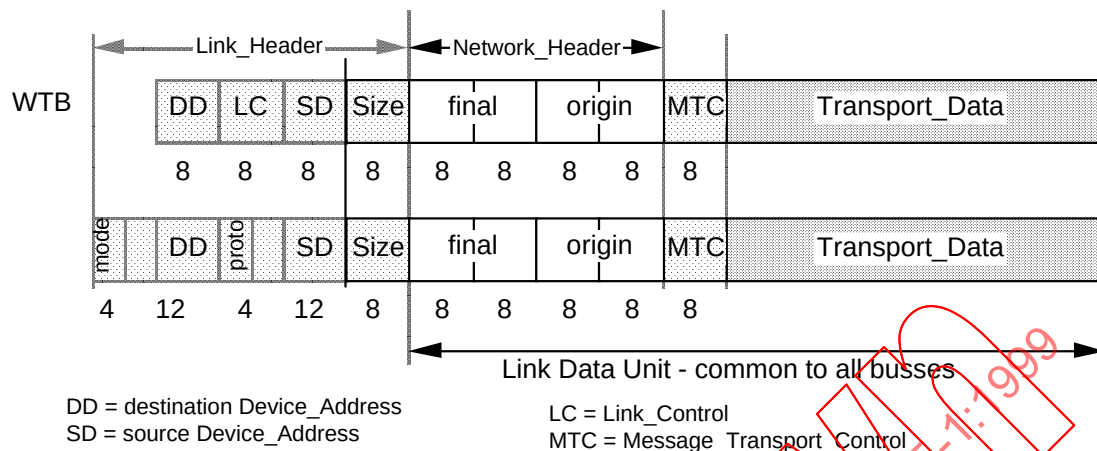


Figure A.45 - WTB and MVB Message_Data frame with Network Addresses

Multi-level hierarchies

The same method as for a System_Address may be used for a User_Address. A Function_Directory may point to a device which is a Router Station. If the specified Function is not implemented by that Router Station, that Station forwards it to another device.

Multicast addressing

The Caller may specify a Group_Address to access a group of Nodes.

The Group_Address is contained in the Final_Address.

On the Train_Bus, the Device_Address used for multicast is the "broadcast" address. The router in each Node decides if its Node belongs to the addressed Group and if this is the case, it forwards the packet to the Station holding the final Function. If there is no selected Function, the router broadcasts the message over the Vehicle_Bus.

On the Vehicle_Bus, the Device_Address is the gateway if the packet is to be broadcast over the Train_Bus only. When the packet is to be broadcast over the Vehicle_Bus, the frame is sent in broadcast mode.

Only Stations of the Vehicle_Bus which implement the Group_Directory may participate in multicast communication.

To this effect, each router maintains a Group_Directory. A Node should subscribe to a particular Group. It does this without consulting other Nodes.

A Group is identified by a 6-bit Group_Address.

Router operation for Message_Data

The router makes no distinction between data packets, acknowledgement and control frames: it treats them all as Packets.

The originating Station sends the Message_Data to its router, specifying the Device_Address of the router in the Link_Header of the frame.

The router receives the Message_Data and analyses the Network_Header to obtain the Network_Address to which the packet should be forwarded to:

- If the final Network_Address specifies a Function_Identifier, the gateway determines via which of its busses and via which Device_Address the final Function can be reached. To this purpose, the gateway holds a Function_Directory;
- If the final Network_Address specifies a Station_Identifier, the gateway deduces the Device_Address. The gateway then forwards the Message_Data to the final device (or to the next router) by specifying the Link_Layer address of that device in the frame. To this purpose, a gateway holds a Station_Directory.

A router does not forward all packets received; some may be addressed to itself (for instance, a Train_Bus Node also supports some Functions).

A router does not participate in the transport protocol: it may not acknowledge packets positively. Positive acknowledgements should come from the Consumer itself.

However, there is a case in which the router intervenes on the protocol. Since a Node_Address may change during message transmission because of a new Inauguration (e.g. due to recovery), inconsistent messages should be rejected since they could arrive to the wrong address. The Node checks that all packets belong to the same Inauguration. To this effect, the Node maintains a counter, the Topo_Counter, which is incremented at each distribution of a Topography. Packets forwarded over the Vehicle_Bus use this Topo_Counter as destination or source Node_Address.

When the Node detects a packet which belongs to a wrong Inauguration, or a packet with no Topo_Counter, it sends a disconnect request with the new Topo_Counter. This allows the sender to know about the new Topo_Counter.

A Vehicle_Bus may be connected to different gateways, each accessed over its own link address.

Example:

A Message_Data transfer from Vehicle_Bus to Vehicle_Bus is shown in Figure A.46.

Function 80 in device 0024 on Node 02 calls Function 93 in device 0033 on Node 05. The transfer is identified by the Network_Address quartet [5,93,2,80], disregarding the addresses of the devices (024 and 033).

The message Link_Layer uses the Function_Directory to identify the source and destination devices. The origin device 024 sends a packet to the gateway as device 020. The gateway decodes the frame. If the final device resides on the same bus (Final Node = 0), the communication is local and the gateway responds as in any other Vehicle_Bus device (the gateway device may additionally implement several Functions).

Since the destination address is Node 05, the gateway generates a Train_Bus frame addressed to Node 05. The gateway inserts its own Node_Address (02) as source device in the frame header.

The packet is forwarded as a datagram on the Train_Bus to gateway 05. The remote gateway decodes the frame and retrieves the destination Function. It inspects the Function_Directory to see where Function 93 resides and finds device 0033.

If the Function were not registered, the Node could broadcast that frame on its Vehicle_Bus, expecting that only one device would respond. In fact, the Function_Directory in the gateway would not be necessary if there were only one Function with that identifier on the Vehicle_Bus: the frames could be broadcast. This, however, cannot be expected in all cases.

The gateway at the Replier side builds a Vehicle_Bus frame, indicating its own address (0030) as the source device and device 0033 as the destination device.

The acknowledgement from device 033 is sent back in the same manner, as a packet.

NOTE - In this example, the Topo_Counter has not been included to simplify the example.

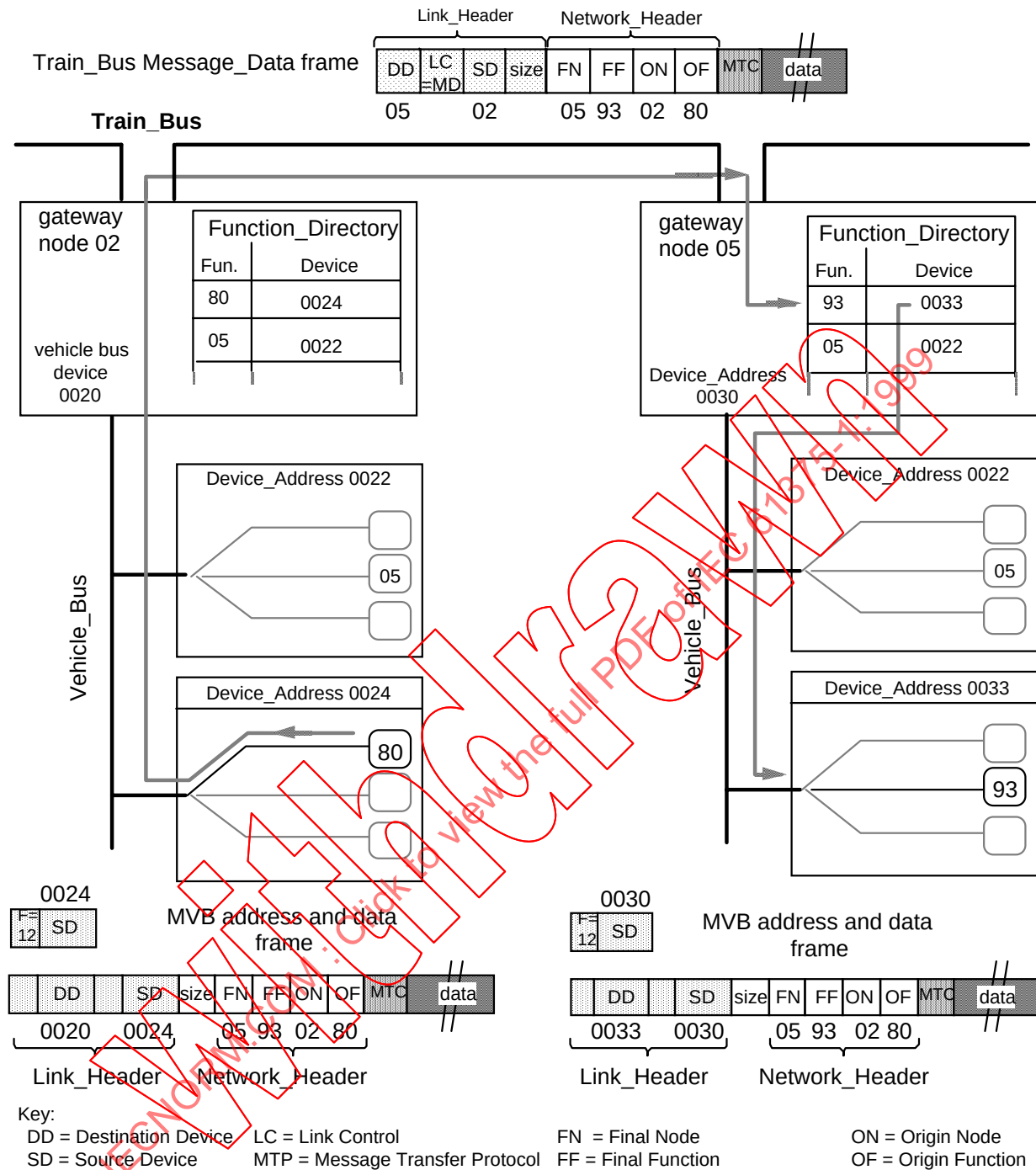


Figure A.46 - Routing messages over the Train_Bus

A.2.3.5 Messages Transport Layer

The Transport_Layer sends one complete message from the Producer to the Consumer. The Transport_Layer segments a lengthy message into a sequence of packets. The Transport Protocol ensures that lost packets are retransmitted and adapts the Producer's transmission rate of packets to the Consumer's rate.

A.2.3.5.1 Involved entities

Since the Train Communication Network transmits only datagrams, the devices where the Functions are implemented cater for the flow control and error recovery. The intermediate devices play no role in the protocol. Thus, Vehicle_Bus devices or the Train_Bus device may execute the Transport Protocol as shown in Figure A.47.

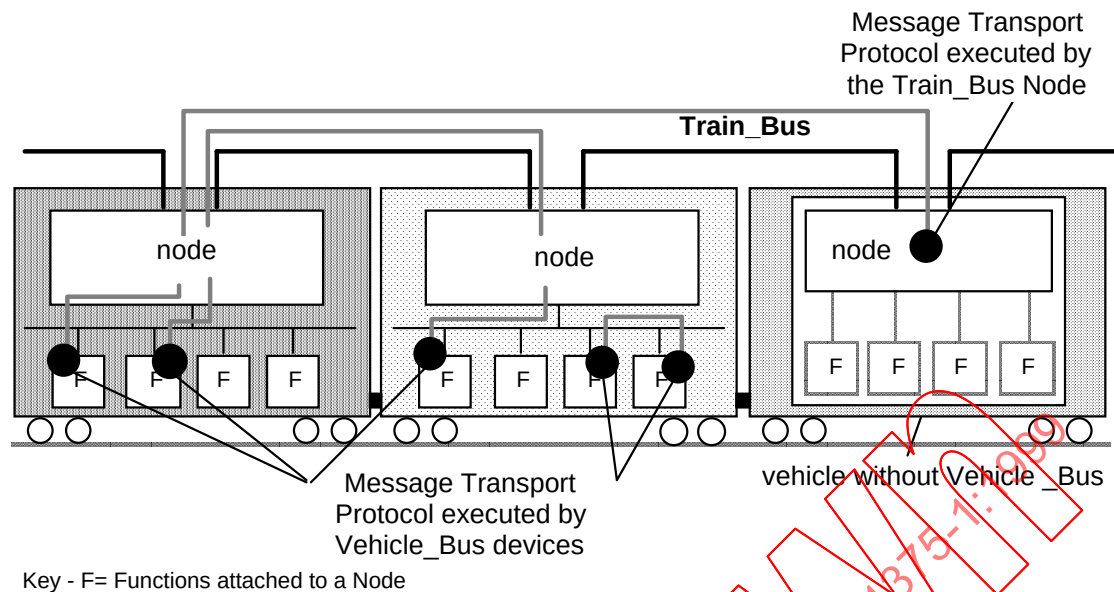


Figure A.47 - Execution of the transport control.

Messages are transmitted as a sequence of packets. The Message Transport Protocol ensures that packets are received in the correct sequence. It provides end-to-end flow control and error recovery.

A.2.3.5.2 Layering

The Transport_Layer builds on the Network_Layer and provides the message transport service to the Application_Layer as shown in Figure A.48.

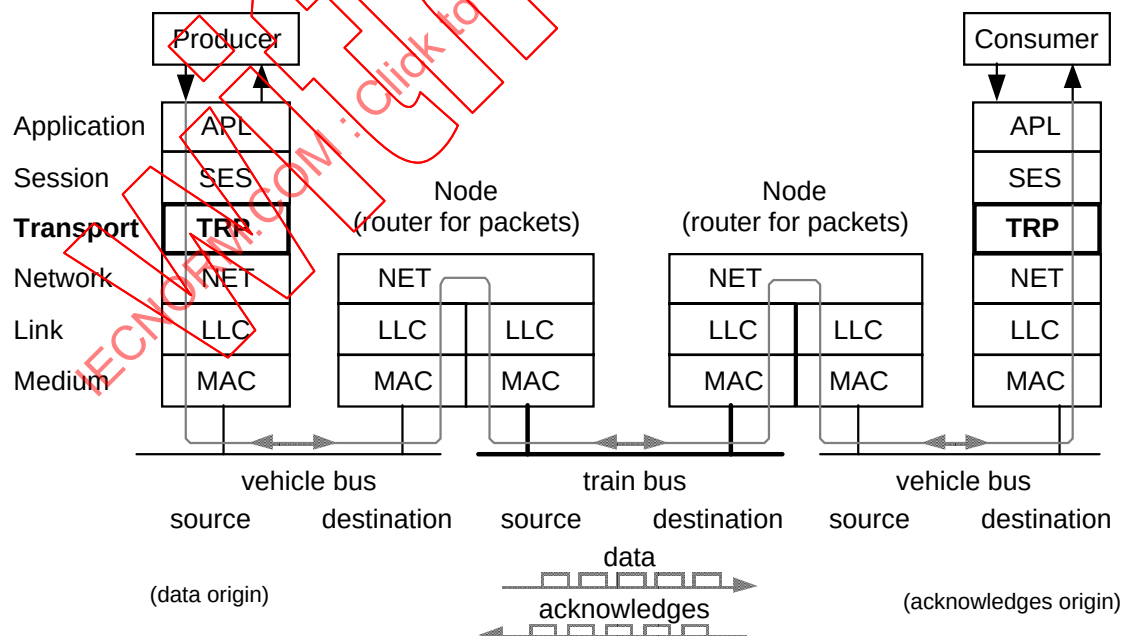


Figure A.48 - Message Transport_Layer in the OSI model

A.2.3.5.3 Sliding window protocol

The Transport_Layer provides flow control and error recovery through a sliding window protocol. This protocol is described in ISO 8073. A simplification of this protocol is used in the Train Communication Network.

The Transport_Layer segments the lengthy messages into small packets, which it numbers in sequence. The sequence number has three bits: the packet numbering is modulo 8, i.e. it continues with 0 after reaching 7.

The Transport_Layer at the Consumer may acknowledge each packet individually or several packets jointly by indicating which sequence number it expects next.

The sliding window protocol allows the sending of packets even if former packets have not yet been acknowledged.

The maximum number of packets which the Producer may send without having received an acknowledgement for any of them is called the "window size". In a simplified implementation, a window size of one may be used, meaning that the packet Producer should receive an acknowledge before it can send the next packet. The maximum size is seven.

Four categories of packets are distinguished, as shown in Figure A.49 for an example with a window size of four:

- 1) packets which have been sent and acknowledged by the receiver;
- 2) packets which have been sent, but not yet acknowledged;
- 3) packets which may be sent without having an acknowledgement for former packets;
- 4) packets which may not be sent, because the window size is exceeded.

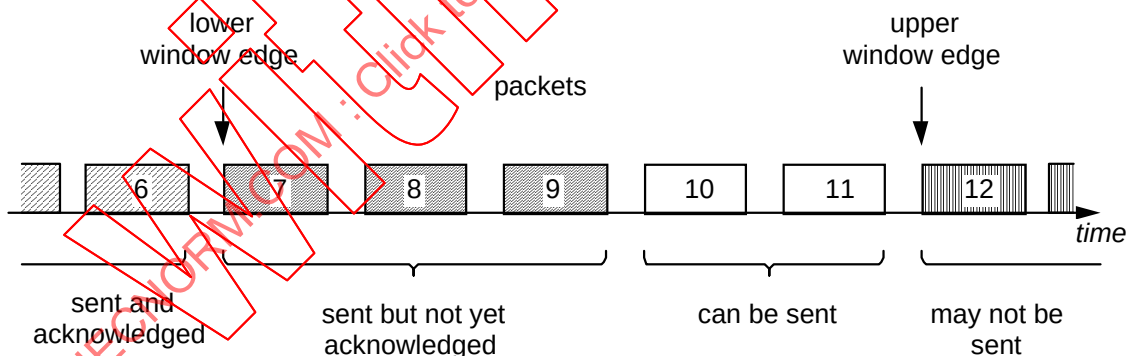


Figure A.49 - Sliding window protocol

When a receiver runs out of buffer space to store the incoming frames, it ceases to send acknowledgement. There is no "receiver not ready" response, since the time-outs may not be modified.

The window size may be varied according to the network load. If the window size is set to 1, a frame may only be sent if the previous one has been acknowledged. The window size may not vary during the message transfer.

The window size is negotiated in the message opening: the packet sender proposes a credit and the packet receiver responds with its credit, which may be lower or the

same as the sender's credit. The credit which the receiver returns will hold for the rest of the message.

The acknowledgement of a packet is also valid for all previous packets. If the Consumer acknowledges packet 8, this is valid for packet 5, 6, and 7 also. This allows the acknowledgement of several data packets with a single acknowledgement packet.

The retransmission strategy is different for the opening frame. The opening frame should be acknowledged before the remaining of the message is sent. When no acknowledgement comes after a complete window has been transmitted, the Producer retransmits the first packet and waits for it to be acknowledged before it may retransmit the others. This avoids congestion at the router.

A.2.3.5.4 Frame exchange

The following Figure A.50 shows a typical frame exchange. In this example, the Consumer acknowledges each packet individually before the Producer may send the next packet (window size =1).

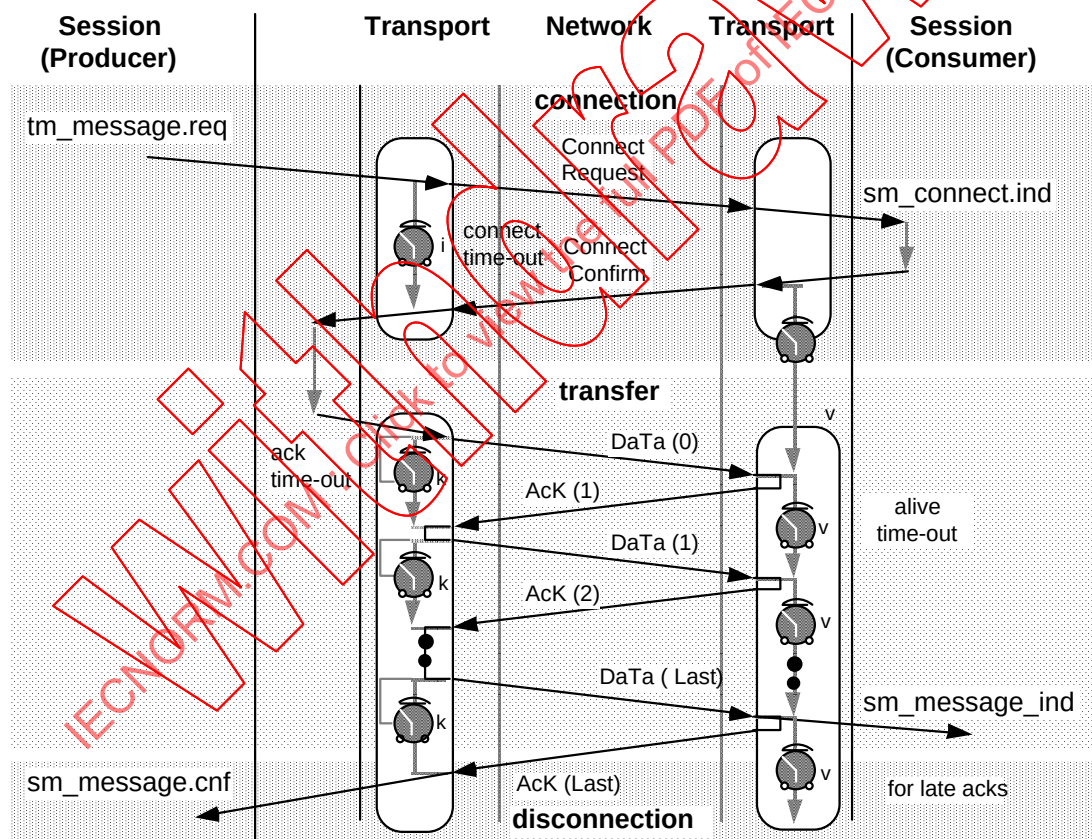


Figure A.50 - Frame exchange at the transport level

The Producer starts the sending of a message with a `Connect_Request` packet, which states the parameters of the `Transport_Layer`: window size, packet size, total size. This first exchange has always a window size of 1, i.e. the Producer waits until it receives either an `Connect_Confirm`, a `Disconnect_Confirm` or times-out.

If the Consumer is correctly addressed and has sufficient resources, it sends a Connect_Confirm packet to accept the message, or otherwise a Disconnect_Request to reject it. The Consumer can condition its acceptance, i.e. specify parameters which are smaller than the proposed parameters of the Producer.

A.2.3.5.5 Credit

Some systems may need to work with a smaller window size than others. To this effect, the window size is negotiated during the call invocation: the Caller proposes a credit and the Replier responds with its credit, which may be lower or the same as the caller's credit. The credit of the Replier rules the next exchange.

For instance, the Producer can propose a window size of seven in the connect request frame. The Consumer accepts the call, but requests a window size of 1. The message exchange should then operate with window size of 1.

During the reply phase, the Consumer indicates its credit and the Producer replies with its own, which may be lower or the same as the Replier's.

The credit may not change afterwards.

A.2.3.5.6 Segmentation

The Transport_Layer performs segmentation of long messages into packets. The frame size is negotiated in the opening frames.

A.2.3.5.7 Packet formats (see Table A.3)

The Message Transport Protocol distinguishes the following types of packets as shown in Figure A.51.

Table A.3 - Packet formats (bus-independent)

packet name	short	origin	parameters
Connect_Request	CR	Producer	requests the reception of a message
Connect_Confirm	CC	Consumer	accepts the reception of a message;
Data	DT	Producer	packet of a message
Positive_Acknowledgement	AK	Consumer	acknowledges reception of packets;
Negative Acknowledgement	NK	Consumer	requests specific retransmission
Disconnect_Request	DR	Producer or Consumer	cancels the transmission
Disconnect_Confirm	DC	Consumer or Producer	confirms the cancellation

The Connect_Request packet indicates:

- the proposed credit (window size);
- the proposed packet size (the length of the Connect_Request frame is imposed by the shortest bus frame type, which is 32 octets on the MVB);